2025 28th International Symposium on Research in Attacks, Intrusions and Defenses (RAID) | 979-8-3315-6603-6/25/$31.00 ©2025 IEEE | DOI: 10.1109/RAID67961.2025.00051

STGraph: Spatio-Temporal Graph Mining for Anomaly Detection in Distributed System Logs

Teng Li*, Shengkai Zhang†, Yebo Feng‡, Jiahua Xu§, Zexu Dang†, Yang Liu¶, Jianfeng Ma†
*†*Xidian University,* ‡¶*Nanyang Technological University,* §*Centre for Blockchain Technologies, University College London*
‡§Exponential Science, **State Key Laboratory of Integrated Services Networks (ISN)*
Email: ‡yebo.feng@ntu.edu.sg

***Abstract*—System logs are crucial sources of information for engineers to analyze and resolve anomalies and faults in large-scale software systems. However, logs on a distributed system are often fragmented, making it challenging to achieve unified processing and comprehension. Traditional methods for log-based anomaly detection often employ machine learning algorithms with a focus on log event counts or log sequences. However, traditional methods fall short of fully leveraging the temporal and spatial structures inherent in distributed system logs, leading to issues of false positives and unstable performance in anomaly detection. In this paper, we propose a novel log anomaly detection method based on the construction of distributed system workflow graphs. This method extracts spatio-temporal information from distributed system logs and constructs event workflow graphs. These graphs accurately reflect the execution of the system and provide more comprehensive support for anomaly detection based on distributed system logs. The experimental results demonstrated that STGraph achieved F1 scores of 0.959, 0.979, and 0.959 on HDFS, BGL, and OpenStack datasets respectively, outperforming LogRobust, PLELog, and NeuralLog by 1.2%-18.6% across precision/recall metrics. Notably, it attained 0.985 recall on BGL and maintained >0.935 F1 scores under 30% noise interference, 21.8% higher than LogRobust.**

***Index Terms*—Log Parsing, Distributed System, Workflow Graph, Anomaly Detection**

I. Introduction

Distributed systems [1] are crucial for supporting numerous web application platforms and are widely used in various sectors, including the internet, finance, manufacturing, military, etc. However, these systems experience more anomalies than traditional centralized systems due to their intrinsic characteristics [2]–[4]. The primary factors contributing to these frequent anomalies include: 1) Partial node failures, where individual nodes within the distributed system malfunction, leading to reduced system availability, data loss, or cascading failures; 2) System overloading, where the processing capacity of the distributed system is overwhelmed by a large number of parallel tasks, resulting in performance degradation or even system crashes; and 3) Data dispersion, where delays or errors in data synchronization between different nodes cause data inconsistency anomalies. Therefore, anomaly detection in distributed systems is vital [5]–[7], defense against potential service disruptions and data corruption, ensuring the robustness and continuity of critical operations.

Corresponding author: Yebo Feng.

In distributed systems, log-based anomaly detection is a highly effective approach to identifying potential problems. This method analyzes log data generated by the system or software during runtime. These logs typically contain comprehensive and detailed descriptions of system behaviors, including activity records, status information, error messages, warnings, and debugging information. Compared to other methods [8]–[10], log-based anomaly detection offers significant advantages. It can provide timely reflections of the operational status of the system or software, enabling rapid detection of anomalies. Furthermore, the comprehensiveness and ease of analysis of log data allow for a thorough understanding of system or software behavior and swift problem localization.

Existing log-based distributed system anomaly detection methods fall primarily into three categories: approaches based on log message counters [11], [12], log event-based [13], [14], and log sequence-based techniques [15], [16]. Log message counter-based methods detect quantitative anomalies by analyzing event frequencies but fail to capture semantic and temporal patterns, limiting their ability to detect complex issues. Log event-based methods analyze log event properties like type and severity, offering more meaningful insights, but often relying on static rules and ignoring temporal relationships. Log-sequence-based methods model the temporal dependencies between events, effectively detecting time-based anomalies, but they are computationally intensive and struggle to integrate event content with temporal dynamics. In general, current log-based anomaly detection methods for distributed systems require further research and improvement to enhance their efficiency, accuracy, and robustness in handling complex log data, adapting to system changes, and minimizing false positives and negatives [17].

Anomalies in large-scale distributed systems often manifest in complex patterns, such as long-term dependencies and periodic changes, which are not directly reflected in logs. However, log data contains temporal correlations, where logs at adjacent time points are relevant. Implicit abnormal information is hidden in the changing relationships between these log events. Current log-based anomaly detection methods struggle with noise, data imbalance, and scalability, leading to high false positives and difficulty detecting rare anomalies. In this study, we propose STGraph, a method that constructs a workflow graph from decentralized logs to detect anomalies.

979-8-3315-6603-6/25/$31.00 ©2025 IEEE
DOI 10.1109/RAID67961.2025.00051

Initially, we employ log parsing techniques [18]–[20] to extract event templates from the logs. Subsequently, by mining the temporal relationships among events, we identify their logical connections, thereby constructing a workflow graph. We embed semantic features, spatial features, and temporal features from log statements into the constructed workflow graph. Using these three features, we design a graph fusion algorithm to complete the fusion operations of subgraphs from different nodes within the distributed system. This ultimately results in the generation of a workflow graph that accurately represents the operational dynamics of the system. By analyzing and processing the workflow graph, a more comprehensive understanding and identification of anomalies can be achieved in distributed system log data, thus effectively performing anomaly detection and troubleshooting.

In the STGraph evaluation, the method demonstrated exceptional anomaly detection efficacy in the HDFS, BGL, and OpenStack Log datasets, with F1 scores of 0.959, 0.979, and 0.959, respectively. When subjected to noise interference such as log data duplication, loss, and sequence disorder, STGraph showed robust resilience, maintaining F1 scores above 0.939 even under high interference conditions. In dealing with various anomaly data injection ratios, STGraph's detection effectiveness remained stable, particularly at higher anomaly ratios, outperforming other methods in precision, recall, and F1 score. Moreover, as the number of nodes in the workflow graph increased, the efficacy of STGraph improved, indicating its effectiveness and precision in handling large-scale log data.

The main contributions of this paper are as follows.

- **Innovative Workflow Graph Construction with Multi-dimensional Feature Embedding.** We have developed a novel approach to constructing a workflow graph that embeds multiple features from interleaved logs. By analyzing the temporal dynamics and logical connections among log events, we create a comprehensive representation that captures temporal, semantic, and spatial characteristics, thereby revealing latent anomalies within the log data.
- **Reduced Influence of Log Noise in Anomaly Detection.** To reduce the influence of noise in the log on the detection results, we filter the nodes of the noise log in the graph by introducing the Top-K pooling mechanism of the Graph Convolutional Network combined with the semantic vector of the nodes of the workflow graph and delete the work generated by the noise log. The flow graph node is used to avoid the misjudgment of noise logs as abnormal logs in the process of abnormal detection, which greatly reduces the false positive rate of abnormal detection.
- **Enhanced Scalability and Versatility of Workflow Graph.** The workflow graph presented in this paper not only accurately reflects the spatial and temporal logical relationships in log events but also demonstrates strong scalability. They are applicable across a wide range of domains, including anomaly detection, fault diagnosis, forensic analysis, performance monitoring, and program analysis and validation, highlighting their extensive utility and adaptability.
- **Extensive Experimental Validation.** We have conducted thorough experiments to validate the accuracy of our method in identifying faulty log sequences. By comparing our approach with three leading quantitative and sequence-based methods on three diverse real-world datasets, we have demonstrated the superior effectiveness and robustness of our proposed technique.

II. Background and Formalization

A. Log Parsing

The logs in distributed systems are typically semistructured, comprising both variables (for example, IP addresses, UUIDs, file paths) that reflect runtime-specific values and constants (for example, fixed log statements) that describe the intended behavior of the program. Each log entry often contains fields such as a timestamp, log level, and host address: the timestamp records the occurrence time of the event, the log level indicates the severity, and the host address identifies the source machine or process. In this study, we employ Drain [18] to parse raw logs into structured event templates, variables, and event types. From the parsed results, we extract time-related features from timestamps, host-related features from IP addresses, and semantic vectors from log templates to capture event characteristics. This structured representation mitigates recognition errors caused by log noise, such as inconsistent formatting, irrelevant entries, and redundant tokens.

Log parsing is crucial for STGraph's workflow graph construction, as it provides the structured event templates for feature extraction and modeling. However, parsing errors—such as misclassified templates or incomplete token extraction—can distort the graph, leading to incorrect or missing event nodes, malformed edges, and fragmented subgraphs. These errors can affect the detection of anomalies by masking true anomalies or generating false positives. For example, misplacing critical events such as "block verification failed" as "unknown error" can mislead the semantic and temporal patterns of the graph, skew the detection performance, and weaken the integrity of the graph.

B. Event Trace

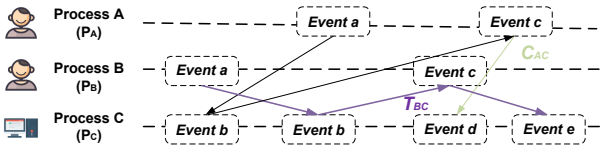


Figure 1: An example of the system trace.

In distributed system log analysis, event traces capture dynamic execution logic across multiple nodes. As shown in Figure 1, a complete system operation typically involves collaborative event sequences from different processes. Our method particularly emphasizes three key attributes of event traces:

Table I: Correspondence between temporal constraints and their linear temporal logic formalizations.

Semantic Description	Time Invariant	LTL Expressions
Strict precedence: whenever e_i occurs, e_j must eventually follow after e_i.	$e_i \rightarrow e_j$	$\mathbf{G}(e_i \rightarrow \mathbf{XF}e_j)$
No precedence: every occurrence of e_i permanently forbids e_j from all future states starting from the next state.	$e_i \nrightarrow e_j$	$\mathbf{G}(e_i \rightarrow \mathbf{XG}(\neg e_j))$
Reverse dependency: if e_j ever occurs, there must exist a prior e_i event, and e_j is forbidden until e_i happens.	$e_i \leftarrow e_j$	$\mathbf{F}e_j \rightarrow (\neg e_j \mathbf{U} e_i)$

Inter-process Correlation: Each trace T_{IJ} represents cross-node collaboration through communication channels C_{IJ}, where events from process P_I may trigger state transitions in P_J.

Temporal Dependency: The partial order relationship $k_i < k_j$ between events e_i and e_j reflects both intra-process sequential execution and interprocess synchronization constraints.

Semantic Consistency: Shared event templates Σ across nodes enable unified representation of distributed behaviors in workflow graphs.

Formally, let Σ be a set of distinct events that appear in all event traces. An trace is a sequence of events, denoted as $< e_1, e_2, ..., e_m >$, where e_i is an event, i.e. $e_i \in \Sigma$ for $1 \leq i \leq m$. Given an event trace $l =< e_1, e_2, ..., e_m >$, for an event e_i, $1 \leq i \leq m$, we call the subsequence $< e_{i+1}, e_{i+2}, ..., e_m >$ as the *postfix* of e_i in l, denoted as $post(e_i)$, and subsequence $< e_1, e_2, ..., e_{i-1} >$ as the *prefix* of e_i in l, denoted as $pre(e_i)$. If $i = 1$, then $pre(e_i)$ is a null sequence, denoted as $\emptyset$. Similarly, when $i = m$, we have $post(e_i) = \emptyset$.

C. Spatio-Temporal Graphs for Distributed Systems

A spatio-temporal graph (STG) is formally defined as a tuple $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathcal{F}_s, \mathcal{F}_q)$ that jointly models spatial topology and temporal dynamics in distributed systems. The node set $\mathcal{V} = \{v_1, ..., v_n\}$ represents physical/virtual entities such as hosts, containers, or microservices, each acting as autonomous execution units. Edges $\mathcal{E} \subseteq \mathcal{V} \times \mathcal{V} \times \mathbb{R}_+$ define time-constrained interactions through directed edges $e_{ij}^{\tau} = (v_i, v_j, \tau)$, where τ is a numerical timestamp indicating that node v_i communicated with v_j at time τ. Spatial features $\mathcal{F}_s : \mathcal{V} \rightarrow \mathbb{R}^{d_s}$ encode multi-granularity topology via *IP_prefix*(v_i) for network locality, *RackID*(v_i) for physical deployment, *HW_type*(v_i) for hardware constraints, and *GNN-Embed*(*connectivity_graph*(v_i)) for learned connectivity patterns. Temporal features $\mathcal{F}_q : \mathcal{E} \rightarrow \mathbb{R}^{d_q}$ capture interaction dynamics using *Time2Vec*(τ) for periodic/decaying time patterns and *LSTM*$(\{\tau_{ij}^k\}_{k=1}^K)$ to model historical interaction sequences $\{\tau_{ij}^k\}$, where k is the number of previous communication timestamps considered. This enables anomaly detection through deviation from learned spatio-temporal invariants.

D. Linear Temporal Logic And Time Invariant

Linear Temporal Logic (LTL) provides a formal framework for specifying temporal invariants in distributed system execution traces [21]. As shown in Figure 1, which captures event dependencies across three processes (A, B, C) with five critical events (a-e), LTL enables rigorous verification of temporal causality through four core temporal operators:

- **Next** ($\mathbf{X}\phi$): Enforces immediate succession where ϕ holds at the next state. For example, the constraint $\mathbf{X}c$ after event a in Process A requires c to become true immediately after any a occurrence.
- **Eventually** ($\mathbf{F}\phi$): Mandates ϕ must hold at some future state. The specification $\mathbf{F}e$ following event b in Process C demands that e eventually occurs after every b instance.
- **Globally** ($\mathbf{G}\phi$): Specifies ϕ must hold perpetually across all states. The invariant $\mathbf{G}(a \rightarrow \mathbf{XF}c)$ for Process A/B sequencing requires that *every* a event must be immediately followed by c (i.e., $\forall a \ \exists c$ with no intermediate states).
- **Until** ($\phi \mathbf{U} \psi$): Defines ϕ must persist continuously *until* ψ becomes true. The progression $c\mathbf{U}e$ in Process C enforces c to remain true continuously until e occurs.

The figure's arrow semantics map directly to LTL formula generation. For the intra-process edge $a \rightarrow c$ in Process A, this translates to $\mathbf{G}(a \rightarrow \mathbf{XF}c)$, where each a occurrence necessitates c in the *immediately subsequent* state ($\nexists$ states between a and c). In contrast, for cross-process dependencies like $c \rightarrow e$ between Process B and C, we derive $\mathbf{G}(c \rightarrow \mathbf{XF}(e \wedge \neg d))$, where $\mathbf{F}$ is used to capture that e will occur at some future state after c, possibly separated by other events, but without any intervening d.

In this work, we specifically employ the time invariant pattern, a subset of LTL, to capture consistent event-ordering relationships across workflows. Temporal invariants establish fundamental constraints on event ordering patterns within distributed system execution. These invariants enable formal verification of workflow logic by capturing necessary precedence relationships between critical system events. As shown in Table I, three temporal relations are formally defined for log analysis: 1) Strict precedence ($e_i \rightarrow e_j$) holds when the occurrence count of e_i equals its appearances before e_j; 2) No precedence ($e_i \nrightarrow e_j$) indicates e_i never precedes e_j; 3) Reverse dependency ($e_i \leftarrow e_j$) occurs when e_j's count matches e_i's preceding appearances of e_j. These temporal constraints are verified through LTL formulations.

E. Non-negative Tensor Decomposition (NTD)

NTD provides a principled framework for extracting interpretable patterns from high-dimensional system logs while preserving inherent spatio-temporal structures. Given a 3-mode log tensor $\mathcal{X} \in \mathbb{R}_+^{T' \times H' \times J'}$ where the modes correspond to temporal slices (T'), host nodes (H'), and event types (J') with:

- Time feature $\tau \in \mathcal{T}$ capturing temporal dynamics.
- Event location feature $h \in \mathcal{H}$ encoding spatial distribution.

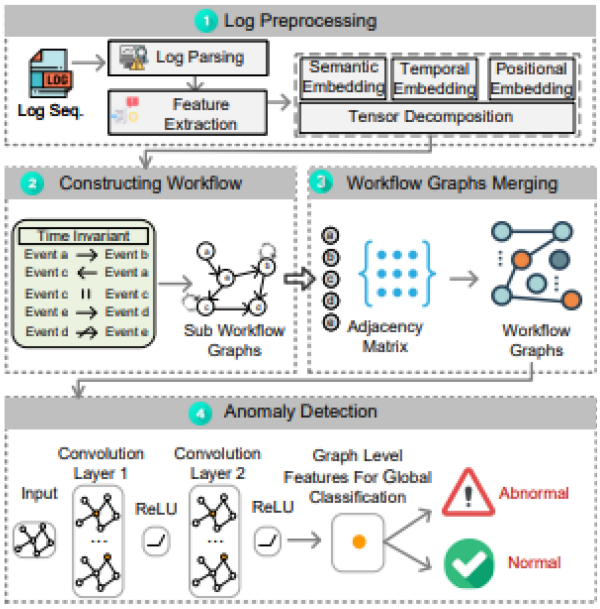


Figure 2: The overview of STGraph.

- Log template semantic feature $j \in \mathcal{J}$ representing event categories.

each element $x_{\tau hj}$ at position (τ, h, j) records multi-dimensional interactions, where $|\mathcal{T}| = T'$, $|\mathcal{H}| = H'$, and $|\mathcal{J}| = J'$ denote the cardinalities of temporal, spatial, and semantic dimensions respectively. NTD factorizes $\mathcal{X}$ into non-negative components:

$$\mathcal{X} \approx \sum_{k=1}^{K} \sum_{l=1}^{L} \mathbf{U}_k^{(\tau)} \circ \mathbf{Z}_{lk}^{(h)} \circ \mathbf{W}_l^{(j)} \tag{1}$$

Here, $\mathbf{U}_k^{(\tau)} = [u_{\tau k}] \in \mathbb{R}_+^{T'}$ denotes the temporal activation pattern of the k-th latent component over time, where τ indexes discrete time steps. $\mathbf{Z}_{lk}^{(h)} = [z_{hlk}] \in \mathbb{R}_+^{H'}$ encodes the participation strength of host h in component k for semantic pattern l, where h indexes distinct hosts in the system. $\mathbf{W}_l^{(j)} = [w_{lj}] \in \mathbb{R}_+^{J'}$ captures the event type distribution for the l-th semantic template, where j indexes distinct event types and w_{lj} represents the probability or weight of event type j appearing in template l. The non-negativity constraint induces a *parts-based* representation, where each component triplet $(\mathbf{U}_k^{(\tau)}, \mathbf{Z}_{lk}^{(h)}, \mathbf{W}_l^{(j)})$ encodes a coherent interaction pattern jointly spanning temporal, spatial, and semantic dimensions.

III. Methodology

A. Overview

The proposed method, shown in Figure 2, constructs workflow graphs from distributed system logs in four sequential phases. In the first phase, unstructured log statements are parsed into structured representations, from which temporal attributes, semantic vectors of log templates, and host identifiers are extracted. To address the high dimensionality and sparsity of these features, non-negative tensor decomposition is applied, yielding compact latent factors that preserve essential spatiotemporal correlations. In the second phase, temporal invariants—representing stable event ordering patterns—are mined from the logs and formalized using LTL, enabling automated verification and guiding the construction of sub-workflow graphs for each node. The third phase merges these subgraphs into a global workflow graph through feature-based graph fusion, ensuring that the integrated representation faithfully captures task execution flows and inter-event dependencies across the distributed system. Finally, in the fourth phase, anomaly detection is performed on the workflow graph using a GCNN with self-attention pooling, which jointly exploits topological structure and node-level features to improve the discrimination between normal and abnormal system behaviors. The complete procedure is summarized in Algorithm 1.

B. Log Preprocessing

Semantic feature extraction involves three steps: preprocessing, vectorization, and aggregation via TF-IDF (Term Frequency-Inverse Document Frequency). We tokenize log events, splitting them into words or subwords while preserving compound terms (e.g., "addStoredBlock"), and remove stop words, numbers, punctuation, and convert words to lowercase, resulting in a filtered sequence $S = [t_1, t_2, \ldots, t_N]$. After preprocessing, we convert S into a semantic vector v. To ensure high recognition accuracy, distinct log events should have low cosine similarity between their semantic vectors, while similar events should have similar vectors to minimize log noise impact. For this, we use the FastText [22] algorithm. FastText is chosen because it captures semantic meaning via subword-level embeddings, making it effective for handling rare or compound terms in log events. Its ability to generate meaningful representations for unseen words and efficiently compute cosine similarity makes it ideal for distinguishing between distinct log events.

To convert the log event statement into a fixed-dimension vector D, we aggregate all N word vectors in the log event LE. First, we calculate the TF-IDF of each word t_i in the log event LE, with $\text{TF-IDF}_{t_i} = \text{TF}_{t_i} \times \ln\left(\frac{N_{t_i}}{\text{DF}_{t_i}}\right)$, where TF_{t_i} is the frequency of t_i in the log statement, N_{t_i} is the total number of logs, and DF_{t_i} is the number of logs containing t_i. The TF-IDF value for word t_i is represented as w_i, and the semantic vector V is calculated as $V = \sum_{i=1}^{n} w_i \cdot v_i$, where v_i is the word vector for t_i and w_i is the TF-IDF weight. TF-IDF is used here to emphasize the importance of words that are frequent within a specific log event but rare across the entire log dataset, helping to capture more distinctive semantic features by amplifying the salience of locally frequent yet globally rare tokens in log vectorization.

To address the high dimensionality and sparsity of extracted log features, we apply NTD to the 3D feature tensor $\mathcal{X} \in \mathbb{R}_+^{T' \times H' \times J'}$, where T', H', and J' denote time segments, host

Algorithm 1 STGraph: Distributed System Anomaly Detection Framework

Require: Distributed system logs $\mathcal{L}$, Rank parameters K, L, Pooling ratio k, Learned classifier parameters W, b

Ensure: Anomaly detection results in y

Phase 1: Log Preprocessing
Parse logs using Drain [18] to extract:
  - Event templates $\mathcal{J}$, Hosts $\mathcal{H}$, Timestamps $\mathcal{T}$
Generate semantic vectors $v_j \in \mathbb{R}^d$ for $j \in \mathcal{J}$:
  a. Tokenize and preprocess log events
  b. Compute TF-IDF weighted FastText embeddings [22]
Construct 3D tensor $\mathcal{X} \in \mathbb{R}^{T' \times H' \times J'}$
Apply NTD decomposition:

$$\mathcal{X} \approx \sum_{p=1}^{K} \sum_{l=1}^{L} \mathbf{U}_p^{(\tau)} \circ \mathbf{Z}_{lp}^{(h)} \circ \mathbf{W}_l^{(j)}$$

Phase 2: Workflow Graph Construction
for each host $h_i \in \mathcal{H}$ **do**
  Mine time invariants as LTL rules to capture event ordering:
    - $\mathbf{G}(e_i \rightarrow \mathbf{XF}e_j)$ for $e_i \rightarrow e_j$ (see Table I)
  Build subgraph $G_i = (V_i, E_i)$ using McScM [23]
    - Validate with counterexample refinement
end for
Phase 3: Graph Fusion
Initialize adjacency matrix $A \leftarrow \mathbf{0}^{m \times m}$
for each subgraph G_j **do**
  Construct A_j via Breadth-First Search (BFS) traversal:

$$a_{oi} = \begin{cases} \lambda_{oi}, & \text{if } v'_o, v'_i \in \mathcal{V}_j \text{ and } e_{oi} \in \mathcal{E}_j, \\ 0, & \text{otherwise} \end{cases}$$

  Aggregate: $A \leftarrow A + A_j$
end for
Phase 4: Anomaly Detection
Apply self-attention graph pooling:

$$Z = \sigma(\widetilde{D}^{-1/2}\widetilde{A}\widetilde{D}^{-1/2}X\Theta_{att})$$

$\text{idx} = \text{top-rank}(Z, \lceil kN \rceil)$
Compute readout:

$$s = \frac{1}{N}\sum_{i=1}^{N} x_i \| \max_{i=1}^{N} x_i$$

Classify: $y = \text{softmax}(Ws + b)$ **return** y

nodes, and unique event templates, respectively. The goal is to approximate $\mathcal{X}$ using the low-rank factors in Equation 1, where $\mathbf{U}_k^{(\tau)}$, $\mathbf{Z}_{lk}^{(h)}$, and $\mathbf{W}_l^{(j)}$ capture temporal activation, host participation, and template distribution.

To ensure the fidelity of the approximation, we minimize the Kullback-Leibler (KL) divergence between the original and reconstructed tensor [24]:

$$\min_{\mathbf{U},\mathbf{Z},\mathbf{W}} \mathcal{D}(\mathcal{X} \,\|\, \mathbf{U}, \mathbf{Z}, \mathbf{W}) \tag{2}$$

where the optimization is subject to non-negativity constraints.

C. Constructing Workflow Graph

A time invariant describes a stable temporal relationship between two events that holds across all normal system executions. Examples include "event e_j must always follow e_i" or "e_j never occurs after e_i." These invariants capture the fundamental temporal logic of the system and serve as constraints to ensure workflow correctness. In our framework, time invariants are mined from logs and formally represented using LTL so that they can be automatically verified and enforced during workflow graph construction.

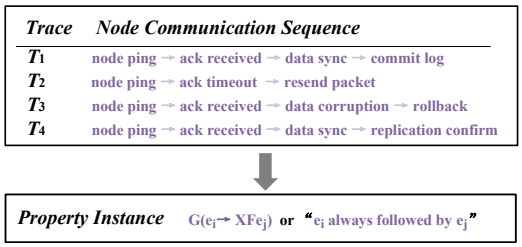


Figure 3: LTL expressions to mine log temporal relationships.

Lemieux [25] proposed a method for mining temporal relationships using LTL expressions, and it is proved that such constants are sufficient to capture critical timing information during system operation. As shown in Figure 3, for a user login log, four trajectories are extracted from it, and the time series relationship we want to mine is selected by setting the attribute type. We choose to mine the $e_i \rightarrow e_j$ time series relationship, so the attribute type is set as $\mathbf{G}(e_i \rightarrow \mathbf{XF}e_j)$, and the attribute instance that conforms to the time sequence relationship can be obtained by mining, that is, $\mathbf{G}(\text{"}guestlogin\text{"} \rightarrow \mathbf{XF}\text{"}authorized\text{"})$, which means that the event *guest login* always appears before the event *authorized*. The specific correspondence between time invariants and LTL expressions is shown in Table I. When mining the time invariants in the log, the LTL attribute types corresponding to three-time invariants can mine the time series relationship between the corresponding events.

In our approach, the model-checking tool McScM [23] is used to test the FSM model obtained in the first step, using this abstract refinement method guided by counterexamples to loop through the model to check whether it satisfies all the time invariants obtained from mining in the previous step, and if there is an unsatisfied state-transfer relation, this counterexample is eliminated until the model satisfies all the time invariants. After correction and refinement, the model can satisfy all the time invariants that we have mined in the logs, and the workflow graph at this point can concisely and accurately represent the execution of the system. At this point, the construction of the workflow graph based on system logs has been completed, and the generated workflow graph can be used in subsequent applications and can play an important role in different application scenarios.

D. Workflow Graphs Merging

The subsystems of the distributed system, the database distribution nodes, are highly autonomous, and the tasks of

the distributed system are carried out separately in each subsystem. Therefore, to understand and monitor the distributed system, it is necessary to integrate the work of each subsystem. First, use the above method to construct the workflow graph of the subsystem, which we call the sub-graph, and then perform the fusion operation of the sub-graphs to obtain the overall workflow graph of the distributed system, which we call the workflow graph. Each subgraph is a directed graph that starts with the $initial$ node and ends with the $terminal$ node. The nodes represent the events executed by the system, the current state, or the information transmitted. The relationship between the nodes is represented by the directed edges and their weights. Then, the fusion algorithm of subgraphs is described as follows:

Let n subgraphs constructed in batches from system sublogs form a tuple $F = \langle F_1, F_2, \ldots, F_n \rangle$. Each subgraph F_j contains k nodes and l edges, with its node set $\mathcal{V}_j = \{v_1, v_2, \ldots, v_k\}$ and edge set $\mathcal{E}_j = \{e_{oi} \mid v_o, v_i \in \mathcal{V}_j, e_{oi} \in F_j\}$. The union of all subgraph nodes of the system can be expressed as $\mathcal{V}_\cup = \mathcal{V}_1 \cup \mathcal{V}_2 \cup \cdots \cup \mathcal{V}_n = \{v'_1, v'_2, \ldots, v'_m\}$, which represents the node set of the general workflow graph and contains m unique nodes. Here, v'_1 denotes the initial node and v'_m denotes the terminal node. Due to the mutual dissimilarity of the elements in $\mathcal{V}_\cup$, nodes that appear repeatedly in multiple subgraphs are represented only once in the union.

The adjacency matrix of the subgraph consists of a vertex table that records the information of each vertex and a two-dimensional array that represents the relationship between each vertex, where the set of vertices $\mathcal{V}_\cup$ of the total graph is used as the vertex table of the adjacency matrix of the subgraph, and the two-dimensional array that represents the relationship of fixed points in the subgraph is constructed as the adjacency matrix at this time, and the adjacency matrix of all subgraphs is a two-dimensional matrix of dimension mm.

Let the adjacency matrix of subgraph F_j be A_j. The element a_{oi} is defined as:

$$a_{oi} = \begin{cases} \lambda_{oi}, & \text{if } v'_o, v'_i \in \mathcal{V}_j \text{ and } e_{oi} \in \mathcal{E}_j, \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

Here, λ_{oi} denotes the weight on the edge e_{oi} from the out-degree node v'_o to the in-degree node v'_i.

Perform breadth-first traversal (BFS) on each subgraph to construct an adjacency matrix for each subgraph.

Finally, it is the construction of a workflow graph for a distributed system. Let the adjacency matrix of the total graph be A, then $A = \sum_{j=1}^{n} A_j$. Based on the adjacency matrix, the GraphViz tool can be used to draw a workflow graph of the workflow, integrating all subsystems under the distributed system.

E. Anomaly Detection

First, label the data. Different from the usual extremely complicated graph node labeling method, this paper directly marks the entire directed graph, which greatly reduces the difficulty of marking and thus reduces the loss of manpower. In this study, we label a total of 1k data points for the pretraining of the GCNN. For each data point, the system workflow graph is compared with the system design diagram: if it matches the design diagram, it is labeled as "normal"; otherwise, it is labeled as "abnormal". The labeled data is then used to train the GCNN model, where the annotations directly influence the model's ability to correctly identify normal and abnormal logs during testing.

After obtaining the batch-labeled workflow graph dataset, the data is pre-trained using a GCNN enhanced by graph pooling based on the self-attention mechanism. This method improves upon traditional GCNNs by incorporating a self-attention mechanism to better handle the classification task of the entire graph. Unlike other graph neural networks, the graph pooling method based on self-attention effectively considers both the overall topology and the individual node characteristics of the graph. This allows the model to perform well on whole-graph classification tasks, leveraging a more comprehensive understanding of the graph's structure.

The attention mechanism has been widely used in various models of deep learning. The self-attention value in our method is computed using graph convolution as follows (see line 26 in Algorithm 1):

$$Z = \sigma\left(\widetilde{D}^{-\frac{1}{2}}\widetilde{A}\widetilde{D}^{-\frac{1}{2}}X\Theta_{att}\right). \tag{4}$$

Among them, $Z \in \mathbb{R}^{N\times 1}$, $\tilde{A} \in \mathbb{R}^{N\times N}$, σ is the activation function, common activation functions are $ReLU$ and $tanh$, etc., $\tilde{A} = A + I_{\mathrm{N}}$ is the undirected graph with self-loop adjacency matrix, I_N is the identity matrix, $\widetilde{D}$ is the degree matrix, X is the F-dimensional input feature of N nodes in the graph, and Θ_{att} is the only parameter of the self-attention graph pooling layer.

The self-attention values Z, derived from graph convolution, encapsulate both the node features and the graph topology. Subsequently, a selection of nodes is performed using the pooling rate $k \in (0, 1]$. Specifically, the first $\lceil kN \rceil$ nodes in Z are chosen via the top-rank function, which returns the indices of nodes ranked within the top $\lceil kN \rceil$. The resulting feature attention mask is denoted as Z_{mask}. The formula for this process is given by: $\mathrm{idx} = \text{top-rank}\left(Z, \lceil kN \rceil\right), \quad Z_{\text{mask}} = Z_{\mathrm{idx}}$, where Z_{mask} retains the **top-$\lceil kN \rceil$ nodes** with highest attention scores, effectively fusing structural saliency and feature relevance.

The pooling operation for the graph is shown: $X' = X_{\mathrm{idx}}, X_{\text{out}} = X' \odot Z_{\text{mask}}, A_{\text{out}} = A_{\mathrm{idx,idx}}$ where X_{idx} is the feature matrix indexed by node, $\odot$ is the bitwise inner product, and $A_{\mathrm{idx,idx}}$ is the adjacency matrix indexed by row and column. X_{out}, A_{out} are the new feature matrix and the corresponding adjacency matrix, respectively.

The Readout layer forms a fixed-size representation by aggregating node features. The specific operation of the readout layer is shown in the following formula [26]:

$$s = \frac{1}{N}\sum_{i=1}^{N} x_i \| \max_{i=1}^{N} x_i, \tag{5}$$

where N is the number of nodes, x_i represents the feature vector of the ith node. The readout layer summarizes and represents the node features in the graph structure as a whole, and obtains the feature representation s of the entire graph.

After obtaining the output s of the readout layer, use it as the input of the graph classification task, map s through a fully connected layer, and use the softmax function to predict the category probability distribution of the graph, the formula is as $y = \text{softmax}(Ws + b)$. Where y represents the probability vector, W represents the weight matrix of the fully connected layer, and b represents the bias vector.

IV. Evaluation

In this section, we perform empirical evaluations to demonstrate the efficacy of our proposed framework STGraph. Specifically, we aim to answer the following research questions (RQs):

- **RQ1**: How does the training data ratio affect the anomaly detection performance of STGraph compared to baseline methods?
- **RQ2**: How efficacious is STGraph in log-based anomaly detection?
- **RQ3**: How much efficacy impact do different types of log noise pollution have on STGraph?
- **RQ4**: How efficacious is STGraph in detecting log anomalies with different imbalanced data?
- **RQ5**: How does the scale of the number of nodes in the workflow graph impact the anomaly detection efficacy of the STGraph model?

A. Datasets

Table II: Statistics of datasets used in evaluations.

Metric	HDFS	BGL	OSL
Time Span	38.7h	214.7 days	27.3h
Data Size	1.47 GB	708.76 MB	1.26 GB
Messages	11,175,629	4,747,963	1,628,503
Templates	48	1,848	2,130
Anomalies	2.93%	10.24%	4.56%

This study compares proposed and baseline methods using three public log datasets: HDFS, OpenStackLog (OSL), and BGL, renowned for their real-world applicability and labeled ground truth. The datasets, sourced from public websites, are summarized in Table II.

B. Baselines and Implementation Details

1) Baselines: To evaluate the effectiveness of our proposed method, we compared STGraph with three state-of-the-art log anomaly detection methods on the aforementioned publicly available log datasets. Specifically, these methods include LogRobust [12], NeuralLog [15], and PLELog [16].

2) Implementation Details: The STGraph graph neural network is implemented using Python 3.8.5 with PyTorch 1.11.0 and PyG 2.04. It consists of a graph encoder and a subsequent 1024-node feedforward network. The AdamW optimizer is utilized with a linearly decaying learning rate from 3×10^{-4} to 1×10^{-9}, a batch size of 64, and a dropout rate of 0.3. Training is guided by a cross-entropy loss function and an early stopping criterion after 20 non-improving iterations, with a maximum of 100 epochs. Each experiment is repeated three times to ensure reliability, with the average reported as the outcome. For the experiments, a window size of 100 log entries is selected, based on the system's log generation frequency and the nature of the anomalies. This window size allows for a good balance between anomaly detection accuracy and computational efficiency. All computations are conducted on an Ubuntu 20.04 server equipped with an AMD Ryzen 3.5GHz CPU, 64GB RAM, and an RTX 3060Ti GPU with 16GB VRAM.

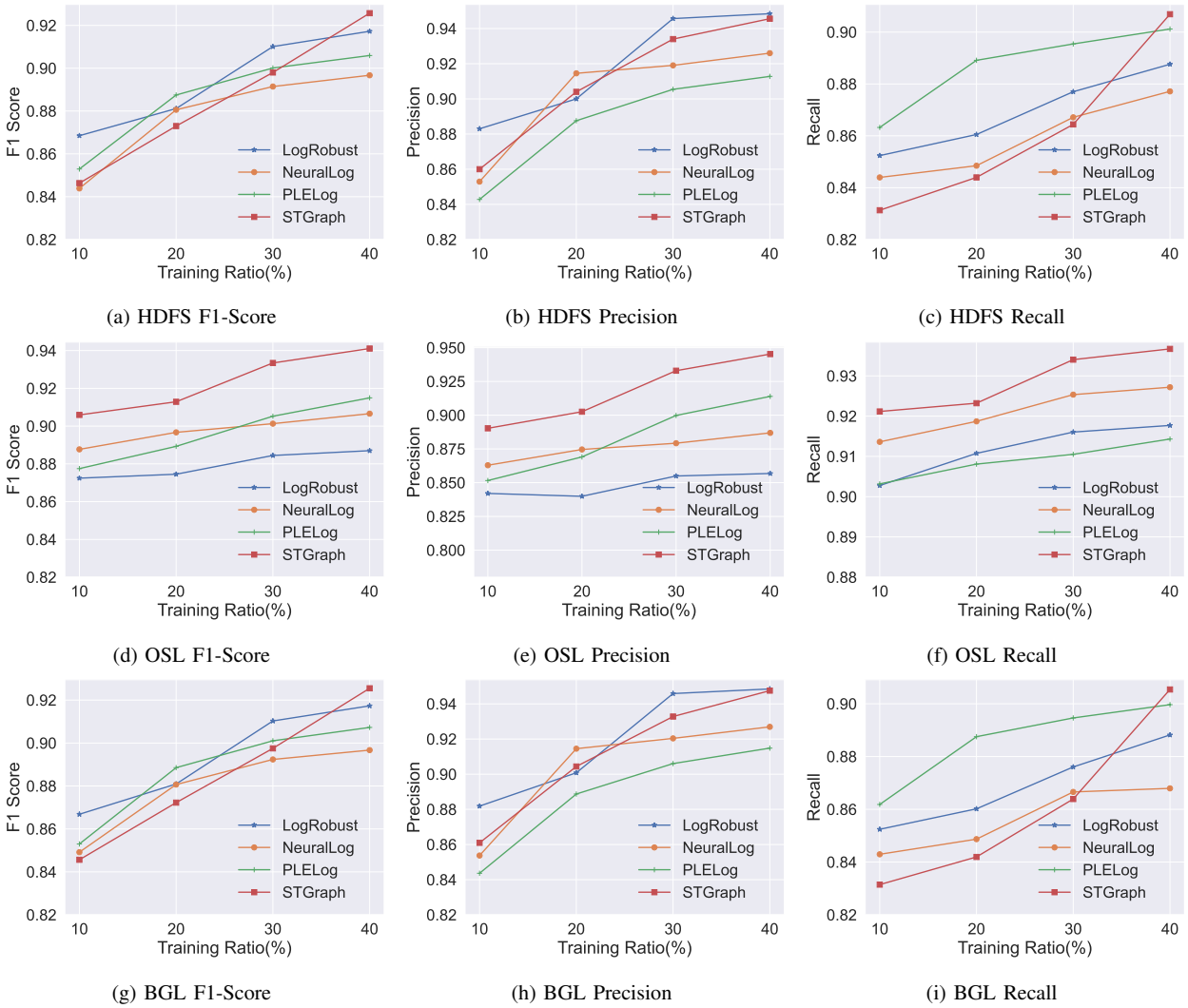
C. RQ1: How does the training data ratio affect the anomaly detection performance of STGraph compared to baseline methods?

As demonstrated in Figure 4a-Figure 4c, STGraph maintains consistent superiority across all metrics on the HDFS dataset. When evaluated at the 40% training ratio shown in Figure 4b, STGraph achieves peak precision of 98%, surpassing LogRobust (96%) and NeuralLog (96%). The recall performance depicted in Figure 4c reveals similar dominance, with STGraph reaching 98% versus 95-96% for baseline methods. This balanced improvement is further evidenced by the F1-score trajectory in Figure 4a, where STGraph sustains a 3-5% advantage across all training ratios.

The OSL results presented in Figure 4d-Figure 4f showcase STGraph's adaptive learning capabilities. As visible in the precision curve of Figure 4e, the method achieves 93.10% precision at 50% training data, outperforming LogRobust's 90.14% and NeuralLog's 89.72%. While NeuralLog leads recall metrics in Figure 4f by 1.78%, STGraph's F1-score of 91.64% in Figure 4d demonstrates a superior balance between precision and recall.

Figure 4h illustrates STGraph's exceptional precision on the BGL dataset, reaching 95.46% at the maximum training ratio. STGraph's F1-score progression demonstrates more stable growth patterns across training increments. The error bars in all subfigures confirm STGraph's lower performance variance compared to baselines, particularly evident in the 20-40% training range.

Finding 1: STGraph demonstrates superior data efficiency and ratio-robust performance, maintaining >93% detection accuracy across all datasets with only half the training data required by conventional methods. This makes it particularly suitable for real-world scenarios with limited labeled logs.



(a) HDFS F1-Score (b) HDFS Precision (c) HDFS Recall

(d) OSL F1-Score (e) OSL Precision (f) OSL Recall

(g) BGL F1-Score (h) BGL Precision (i) BGL Recall

Figure 4: Performance comparison on HDFS dataset under varying training ratios.

D. RQ2: How efficacious is STGraph in log-based anomaly detection?

An evaluation experiment was conducted to assess the performance of STGraph using three public log datasets. The training and testing sets were divided randomly, with oversampling applied to achieve a 20% anomaly rate in the training data. A fixed window size of 200 logs was set for input data. Experimental results, presented in Table III, indicate STGraph's dominance over LogRobust, PLELog, and NeuralLog in F1-score across all datasets.

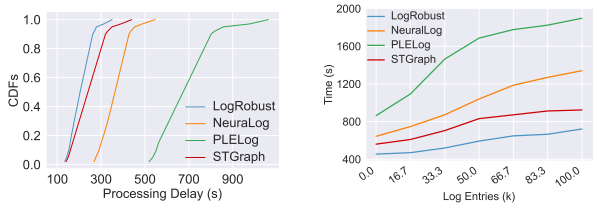
STGraph surpasses LogRobust by leveraging word frequency information for model construction, achieving an F1-score of 0.959 on HDFS compared to LogRobust's 0.947, and significantly improving from LogRobust's 0.795 to 0.979 on the BGL dataset. This highlights STGraph's robustness against unstable log data. In comparison with PLELog, which employs semi-supervised learning and probabilistic label estimation, STGraph consistently performs better, suggesting that PLELog's performance may be influenced by the effectiveness of its clustering algorithm. PLELog's F1-scores on BGL and OSL datasets are 0.956 and 0.954, respectively, yet slightly lower than STGraph. NeuralLog, a deep learning approach using pre-trained language models for semantic analysis, falls short of STGraph's performance. For instance, on the OSL dataset, STGraph's F1-score of 0.959 is notably higher than NeuralLog's 0.846, indicating the practicality and effectiveness of STGraph's word frequency-based method for anomaly detection.

STGraph has proven to be effective in detecting anomalies in log data, demonstrating the ability to learn from normal patterns and identify anomalies accurately, even amidst data instability. The graph-based structure of STGraph, which mod-

Table III: Efficacy results of different approaches on different datasets.

Dataset	Metric	STGraph	LogRobust	PLELog	NeuralLog
HDFS	Pre	0.963	**0.972**	0.950	0.925
	Rec	**0.955**	0.923	0.937	0.947
	F1	**0.959**	0.947	0.944	0.936
	Acc	**0.972**	0.960	0.970	0.935
	Spe	**0.984**	0.950	0.960	0.965
BGL	Pre	**0.973**	0.683	0.941	0.946
	Rec	**0.985**	0.952	0.972	0.967
	F1	**0.979**	0.795	0.956	0.956
	Acc	**0.976**	0.950	0.970	0.955
	Spe	**0.985**	0.940	0.965	0.970
OSL	Pre	**0.972**	0.851	0.958	0.862
	Rec	0.946	0.742	**0.949**	0.831
	F1	**0.959**	0.793	0.954	0.846
	Acc	**0.965**	0.940	0.960	0.951
	Spe	**0.975**	0.935	0.955	0.960

els complex relationships among log events, provides a nuanced representation of system behavior. It exceeds sequence-based methods by characterizing complex event dependencies through various types of edges.



(a) Cumulative Distribution Function for anomaly detection.

(b) Runtime performance with varying log sizes.

Figure 5: Efficiency analysis of STGraph.

We analyze the anomaly detection effect and performance of each method in log processing. Figure 5a shows the cumulative distribution function curve of anomaly detection after collecting logs for a certain period and shows the accuracy and effect of anomaly detection under different log data amounts. The STGraph method has the best performance among all comparison methods, showing its efficiency and accuracy in processing log data of different sizes. Figure 5b shows the changes in system running time under different numbers of logs. The running time of STGraph ranks second, which shows that this method can maintain relatively superior computing efficiency while ensuring high accuracy.

To evaluate STGraph's deployability in large-scale log processing, we compared its runtime performance with LogRobust, PLELog, and NeuralLog, as shown in Table IV. The results reveal that, while all methods exhibit roughly linear runtime growth with increasing log volumes, LogRobust has the lowest runtime but sacrifices detection accuracy. NeuralLog, using LSTM-based models, shows the highest runtime overhead. In contrast, STGraph consistently ranks second, with a runtime that scales efficiently, even when processing 100K logs. Despite the additional graph construction and GCN-based inference, STGraph maintains competitive performance, offering a favorable balance between detection effectiveness and computational cost, making it well-suited for large-scale, real-world deployment.

Table IV: Performance comparison of different log parsing approaches (in seconds).

Log Entries (k)	LogRobust	NeuralLog	PLELog	STGraph
16.7	454.58	643.79	863.16	560.14
33.3	469.66	747.19	1093.29	609.68
50.0	519.21	871.77	1463.09	704.11
66.7	592.45	1039.44	1686.76	831.20
83.3	648.10	1185.92	1777.23	871.77
91.6	665.33	1269.57	1824.27	912.70
100.0	721.34	1340.66	1897.51	923.47

Finding 2: STGraph excels in log data anomaly detection by learning from normal patterns and accurately identifying anomalies while maintaining high efficiency.

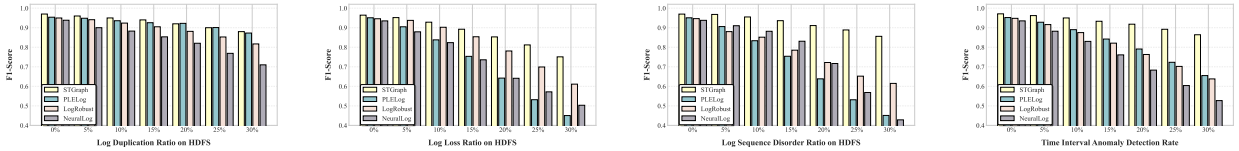
E. RQ3: How much efficacy impact do different types of log noise pollution have on STGraph?

1) Log Duplication Experiment: To evaluate the performance of STGraph in scenarios with log data duplication, we designed a data replication experiment. We randomly selected a portion of log records from the original log data, duplicated them at certain ratios, and mixed them into the original log data. Through this approach, we constructed log datasets with varying duplication rates.

The experimental results are shown in Figure 6a. The results demonstrate that STGraph achieves the best performance with an F1 score of 0.982 when no log data duplication exists. As the duplication rate increases, STGraph's performance slightly degrades but still surpasses the baseline method. When the duplication rate reaches 30%, STGraph's F1 score remains above 0.935, showcasing its strong robustness. This is primarily attributed to STGraph's utilization of log event relationship graphs for diagnosis, enabling it to effectively address noise issues within log data.

2) Log Loss Experiment: The STGraph method's diagnostic performance in the face of log loss was evaluated through experiments that introduced varying loss rates. The experimental results are shown in Figure 6b. Even with a 30% loss rate, STGraph maintains an F1 score above 0.75, illustrating its robustness due to its log event relationship graph-based diagnosis.

In comparison, the baseline method, based on fixed pattern matching, is more susceptible to log loss and experiences a faster performance decline. This suggests that such methods are sensitive to missing data and struggle to effectively address log quality problems that arise in real-world systems.



(a) Log Duplication Ratio. (b) Log Loss Ratio. (c) Log Sequence Disorder Ratio. (d) Time Interval Anomaly Rate.

Figure 6: F1-score comparison on HDFS dataset with log duplication rates, log loss rates, and log sequence disorder rates.

Table V: Comparison of the efficacy of different approaches in processing varying abnormal data ratios on the HDFS dataset.

Method	5%				10%				15%				20%			
	Pre	Rec	F1	Acc/Spe	Pre	Rec	F1	Acc/Spe	Pre	Rec	F1	Acc/Spe	Pre	Rec	F1	Acc/Spe
LogRobust	0.787	0.936	0.855	0.871/0.862	0.850	0.941	0.893	0.889/0.873	**0.945**	0.929	0.937	0.910/0.900	**0.972**	0.923	0.947	0.930/0.927
PLELog	0.703	0.923	0.798	0.864/0.852	0.782	0.933	0.851	0.873/0.868	0.882	0.941	0.911	0.905/0.894	0.950	0.937	0.944	0.920/0.915
NeuralLog	0.871	0.954	0.911	0.890/0.886	0.906	0.965	0.935	0.901/0.898	0.922	0.936	0.929	0.929/0.910	0.925	0.947	0.936	0.940/0.938
STGraph	**0.905**	**0.983**	**0.942**	**0.926/0.917**	**0.913**	**0.988**	**0.949**	**0.931/0.923**	0.932	**0.947**	**0.939**	**0.943/0.935**	0.963	**0.955**	**0.959**	**0.960/0.955**

3) Log Sequence Disorder Experiment: The STGraph method was subjected to an experiment simulating log sequence disorder by shuffling records, reflecting real-world out-of-order log occurrences. The experimental results are shown in Figure 6c. Despite the increased disorder, STGraph sustains superior performance, with an F1 score above 0.86 even at a 30% disorder rate, underscoring its robustness. This robustness is attributed to STGraph's log event relationship graph diagnosis, which adeptly manages sequence inconsistencies, unlike pattern-matching baselines that decline rapidly with disorder, revealing their limited adaptability to real-world log data variability.

4) Time Interval Anomaly Experiment: To evaluate STGraph's capability in handling temporal pattern distortions, we conducted controlled experiments by injecting timestamp-based anomalies into HDFS logs. These anomalies simulate two real-world scenarios: 1) Burst event sequences with abnormally short intervals (0.1s), and 2) Suspiciously long silent periods (>1000s) between consecutive logs. The anomaly injection ratio was systematically varied from 0% to 30% to simulate escalating temporal distortions.

The experimental results are shown in Figure 6d. STGraph demonstrates superior temporal robustness, achieving a baseline F1-score of 0.971 under normal conditions. When the anomaly ratio reaches 30%, it still maintains 86.4% detection accuracy, with only a 10.7% decrease in performance. NeuralLog experiences the sharpest decline (0.935 to 0.527), since its sliding window mechanism cannot effectively capture global temporal contexts. LogRobust records a 31% reduction in accuracy (0.948 to 0.638) due to its reliance on fixed temporal statistics. Moreover, once the anomaly ratio exceeds 15%, STGraph consistently outperforms PLELog by 9.1 to 20.9 percentage points, highlighting the necessity of incorporating joint spatio-temporal modeling.

5) Log Ablation Experiment: Table VI summarizes the impact of removing key components from the STGraph on its performance metrics. The full STGraph configuration achieves the highest precision, recall, F1 score, accuracy, and specificity. However, excluding specific components results in noticeable performance degradation. Removing Log Parsing reduces the F1 score by 0.138, emphasizing its critical role in identifying log patterns. The absence of Feature Extraction leads to a more significant F1 score drop of 0.193, highlighting the importance of extracting meaningful features for effective diagnostics. Similarly, omitting Tensor Decomposition decreases the F1 score by 0.159, showcasing its necessity for managing high-dimensional log data and enhancing diagnostic accuracy. These results demonstrate the integral contributions of each component to the robustness and effectiveness.

Table VI: Ablation study results for this method. The down arrow (↓) indicates the missing accuracy of this part of the method.

Metric	Ablated Part			
	None	Parsing	Feature Extraction	Tensor Decomposition
Precision	0.912	0.774 (↓0.138)	0.729 (↓0.183)	0.726 (↓0.186)
Recall	0.891	0.749 (↓0.142)	0.688 (↓0.203)	0.755 (↓0.136)
F1	0.901	0.761 (↓0.140)	0.708 (↓0.193)	0.742 (↓0.159)
Accuracy	0.910	0.763 (↓0.147)	0.724 (↓0.186)	0.750 (↓0.160)
Specificity	0.895	0.759 (↓0.136)	0.711 (↓0.184)	0.751 (↓0.144)

Finding 3: STGraph exhibits remarkable robustness in log anomaly detection, maintaining high-performance metrics even in the presence of log duplication, loss, and sequence disorder.

F. RQ4: How efficacious is STGraph in detecting log anomalies with different imbalanced data?

In this section, we analyze the comparative efficacy of the STGraph method against other prominent log anomaly detection methods—LogRobust, PLELog, and NeuralLog—on the HDFS dataset. The experimental results are shown in Table V. The superior performance of STGraph can be attributed to its sophisticated approach in leveraging the spatio-temporal dynamics of logs, which enables a more nuanced and accurate

detection of anomalies. This method's ability to integrate and harmonize the asynchronous and dispersed nature of logs from distributed systems provides a distinct advantage over other methods, which often struggle with the complex interplay of log data at higher anomaly rates.

The comparative analysis reveals that while STGraph maintains high precision and recall, other methods, such as PLELog, exhibit limitations, particularly at lower anomaly rates, suggesting a lack of sensitivity to subtle deviations in log data. This underscores the importance of a method's capacity to discern anomalies amidst a sea of normal log entries, a capability where STGraph excels.

In essence, STGraph's success lies in its comprehensive handling of the multidimensional aspects of log data, offering a robust framework for anomaly detection in distributed systems. The findings highlight the need for methods that can adeptly navigate the intricacies of log data to ensure reliable and precise anomaly detection.

Finding 4: STGraph's superior efficacy in log anomaly detection underscores its capability to adeptly leverage the spatiotemporal complexity of log data, providing a robust framework that outperforms existing methods, especially at higher anomaly rates.

G. RQ5: How does the scale of the number of nodes in the workflow graph impact the anomaly detection efficacy of the STGraph model?

Table VII: Performance metrics for different groups.

Metric	100	200	300	400	500
Precision	0.931	0.963	0.969	0.976	**0.982**
Recall	0.962	0.955	0.952	**0.964**	0.961
F1	0.946	0.959	0.960	0.970	**0.971**
Accuracy	0.935	0.960	0.970	0.975	**0.980**
Specificity	0.955	0.950	0.955	0.960	**0.965**

In our experiments, we investigated the anomaly detection performance of the STGraph model across varying scales of workflow graph nodes, using real Hadoop system log data. Constructing workflow graphs with nodes ranging from 100 to 500, we applied GCNN for anomaly detection with a fixed set of parameters. The experimental results are shown in Table VII. The results indicate that as graph scale increases, so does the STGraph model's precision, recall, and F1-score, suggesting that larger graphs provide a richer context for more effective anomaly pattern recognition. This is because additional nodes contribute more data points and relationships, which in turn provide a denser and more comprehensive representation of the system's behavior. This density allows the GCNN to capture complex patterns and dependencies that may be indicative of anomalies, thus improving the detection accuracy.

Finding 5: The STGraph model excels in anomaly detection with larger workflow graphs, leveraging increased node density for a nuanced understanding of normalcy and anomaly patterns.

H. Anomaly Detection Instance.

We use the HDFS log topology graph generated by RT-Graph as the baseline prototype. This topology graph captures the full execution workflow of the HDFS distributed file system. To make abnormal behavior more identifiable, we simplify the complex network structure by retaining only subgraphs that contain key abnormal patterns. This enables a direct comparison of the detection capabilities of STGraph against alternative methods such as LogRobust, PLELog, and NeuralLog.

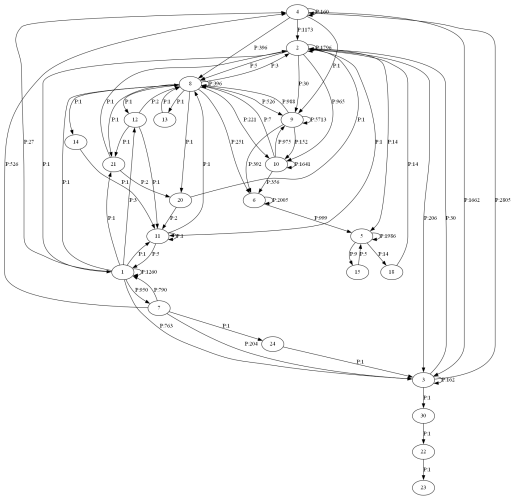


Figure 7: Local topology of HDFS log workflow Diagram (Simplified Schematic diagram).

The simplified local topology in Figure 7 represents a workflow graph constructed from standardized HDFS log events. Each node corresponds to a specific system operation (e.g., block replication, file deletion) and contains a unique event ID assigned after log parsing. Directed edges indicate the immediate succession of events within the same workflow, and each edge is labeled with a weight in the form P:N, where P is the path identifier and N is the number of times that event sequence appears in the logs (e.g., P:396 means the sequence occurred 396 times).

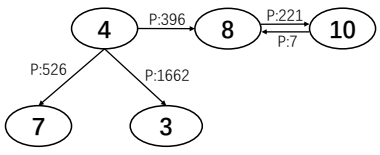


Figure 8: Abnormal subgraph patterns of malicious block injection attacks.

From this topology, we extract high-risk abnormal subgraph patterns, as illustrated in Figure 8. These patterns correspond to malicious block injection attacks that can disrupt normal HDFS operations. The mapping between node types and their corresponding system operations is summarized in Table VIII, which defines the semantic role of each node in the workflow graph.

Table VIII: Mapping between HDFS log events and graph nodes.

Node ID	Semantic Meaning
Node 4	`Received block` *(Data block reception event)*
Node 8	`PacketResponder: block verification failed` *(Critical security failure event)*
Node 10	`Starting thread` *(Daemon thread initialization)*
Node 3	`Exception in` *(System exception reporting)*
Node 7	`Delete block` *(Storage block deletion operation)*

To further demonstrate the practical effectiveness of ST-Graph in detecting complex anomalies within distributed systems, we conducted a detailed case study based on the HDFS dataset. In this case, STGraph successfully identified a critical anomaly pattern related to malicious block injection attacks, which remained undetected by baseline approaches including LogRobust, PLELog, and NeuralLog. The anomaly was reflected in the constructed workflow graph as a distinct abnormal subgraph, characterized by irregular hub connectivity, anomalous edge weights, and temporal order violations.

The extracted workflow subgraph revealed a suspicious execution chain originating from the event [Node 4] Received block, which subsequently branched into multiple event paths. Notably, one of the dominant paths directed towards [Node 8] Block verification failed with a high occurrence frequency of 396, indicating its frequent involvement in system operations. From Node 8, two divergent event flows were observed: one leading to [Node 7] Block deletion with a frequency of 526, which conformed to expected behavior, and another anomalous path targeting [Node 3] Exception thrown, occurring 1662 times. This latter edge exhibited a frequency approximately 3.7 times higher than the baseline established from normal system executions, signaling potential malicious activity.

A deeper inspection uncovered further evidence of anomalous behavior. In legitimate system operations, thread initialization events, such as [Node 10] Thread started, should logically precede block verification failures, following the designed execution flow of HDFS. However, the observed sequence in this anomaly case showed a reversed dependency—Node 8 preceding Node 10—indicating a temporal violation inconsistent with the system's operational semantics.

While this anomaly manifested clearly within the graph structure, traditional methods failed to capture it due to inherent limitations in their detection strategies. LogRobust, relying solely on event frequency statistics, misclassified the high-frequency occurrence of Node 8 as benign retry behavior, and thus overlooked abnormal branching. PLELog, which applies clustering on sequential event patterns, failed to distinguish the abnormal sub-path from Node 4 to Node 3, as its model could not account for the branching and parallelism inherent in workflow graphs.

V. Related Work

Anomaly detection in log data is a critical area of research, particularly for maintaining the reliability and performance of large-scale distributed systems [31], [32]. As summarized in Table VII, this section reviews the state-of-the-art in log-based anomaly detection, categorized into three main groups: log message counter-based anomaly detection, log event-based anomaly detection, and log sequence-based anomaly detection.

A. Log Message Counters-Based Anomaly Detection

Log message counter-based methods represent log data as vectors, focusing on the frequency of specific log events over time. These methods analyze the statistical distribution of log message occurrences, allowing for the detection of anomalies based on changes in event counts. *LogRobust* [12] is an example of such an approach, which enhances the robustness of log-based anomaly detection by leveraging noise-robust techniques to better handle the inherent noise in log data. Another example is *LogUAD* [27], which employs word2vec embeddings to convert log messages into numerical vectors and uses clustering techniques to identify anomalies. These methods are effective in detecting quantitative anomalies, but they may struggle with capturing the full semantic context of the log events, which can limit their ability to detect more complex, context-dependent anomalies.

B. Log Event-Based Anomaly Detection

Log event-based methods [28], [33]–[38] focus on detecting anomalies by analyzing individual log events and their properties, such as event type, severity, and frequency. These methods often rely on predefined rules or statistical models to identify events that deviate from expected patterns. *LogAnomaly* [13] combines sequence modeling and clustering techniques to detect both sequential and quantitative anomalies in unstructured logs. *LogCluster* [14] is another notable method, which uses clustering to identify patterns in log events and detect abnormal behavior. This approach is effective at grouping similar events, making it easier to identify outliers that deviate from typical event patterns. These methods are particularly useful for detecting anomalies related to specific log event types, such as errors or system misconfigurations, but may struggle with handling temporal dependencies or complex patterns in log data.

C. Log Sequence-Based Anomaly Detection

Log sequence-based methods [29], [39]–[44] take a sequence of log events as input. They typically use various deep learning models to learn sequential patterns in log sequences

Table IX: Summary of log-based anomaly detection methods.

Category	Methods	Key Technology	Strength	Limitation
Log Message Counters-Based	LogRobust [12]	Frequency vectors	Noise resistance	Semantic context loss
	LogUAD [27]	Word2vec embeddings	Unsupervised learning	Context unawareness
Log Event-Based	LogAnomaly [13]	Sequence modeling	Hybrid detection	Temporal blindness
	LogCluster [14]	Pattern clustering	Outlier detection	Static thresholds
	SwissLog [28]	Workflow parsing	Temporal modeling	High overhead
Log Sequence-Based	DeepLog [11]	LSTM networks	Temporal dependency	Data hunger
	LayerLog [29]	Hierarchical semantics	Context awareness	Interpretability loss
	nLSLog [30]	NMF technology	Dimension reduction	Sequence fragmentation

for anomaly detection, showing impressive performance. *nLSLog* [30] utilizes non-negative matrix factorization (NMF) technology to process log sequence data, discovering hidden patterns in the logs to detect abnormal behaviors. *LayerLog* [29] proposes a new framework for anomaly detection of log sequences based on hierarchical semantics. One notable work in this category is *DeepLog* [11], which uses a long short-term memory network to learn the patterns of normal system behaviors from historical log data. The model then identifies anomalies by detecting deviations from these learned patterns. Such methods require extensive labeled data, which can be challenging to obtain in practice.

In our approach, we extract spatio-temporal information from distributed system logs to construct event workflow graphs. These graphs accurately reflect system execution and provide more comprehensive support for log-based anomaly detection. We embed semantic, spatial, and temporal features from log statements into the constructed workflow graph, which ultimately generates a graph that accurately represents the operational dynamics of the system. Compared to existing methods, our approach not only improves the accuracy of anomaly detection but also significantly reduces the false positive rate. This multi-feature fusion-based workflow graph construction method offers a more comprehensive and stable solution for anomaly detection in distributed systems.

VI. Conclusion

In this paper, we presented a novel method for anomaly detection in distributed systems based on constructing workflow graphs from log data. Our approach addresses the challenges posed by fragmented and interleaved logs by extracting temporal, semantic, and spatial features to build comprehensive workflow graphs. These graphs encapsulate the dynamic execution and relationships within the system, providing a robust foundation for identifying anomalies.

Our contributions include: 1) a technique for constructing comprehensive workflow graphs from dispersed logs; 2) a graph fusion algorithm for integrating subgraphs from different nodes; 3) validation of our graphs' scalability and versatility in anomaly detection, fault diagnosis, and performance analytics; and 4) comprehensive experiments demonstrating our method's superior accuracy and effectiveness. Our method shows significant improvements over traditional log-based anomaly detection techniques by leveraging spatio-temporal structures in distributed system logs. Extensive experiments on the HDFS, BGL, and OSL datasets demonstrate superior F1 scores of 0.959, 0.976, and 0.959 respectively, outperforming state-of-the-art methods.

Acknowledgments

This research is funded by the National Key Research and Development Program of China (2023YFB2904000), Natural Science Basic Research Program of Shaanxi (No. 2025JC-JCQN-073), National Natural Science Foundation of China under Grant (No. 62272370), Young Elite Scientists Sponsorship Program by CAST (2022QNRC001), the China 111Project (No.B16037), Qinchuangyuan Scientist + Engineer Team Program of Shaanxi (No. 2024QCY-KXJ-149), Songshan Laboratory (No. 241110210200), Open Foundation of Key Laboratory of Cyberspace Security, Ministry of Education of China (No.KLCS20240405) and the Fundamental Research Funds for the Central Universities (QTZX23071), the National Research Foundation, Singapore, and DSO National Laboratories under the AI Singapore Programme (AISG Award No: AISG2-GC-2023-008), the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN), the National Research Foundation, Prime Minister's Office, Singapore under its Campus for Research Excellence and Technological Enterprise (CREATE) programme, and UK Centre for Blockchain Technologies through Ripple's University Blockchain Research Initiative (UBRI) [45].

References

[1] J. Wu, *Distributed system design*. CRC press, 2017.
[2] R. Anderson, *Security engineering: a guide to building dependable distributed systems*. John Wiley & Sons, 2020.
[3] D. Palko, T. Babenko, A. Bigdan, N. Kiktev, T. Hutsol, M. Kuboń, H. Hnatiienko, S. Tabor, O. Gorbovy, and A. Borusiewicz, "Cyber security risk modeling in distributed information systems," *Applied Sciences*, vol. 13, no. 4, p. 2393, 2023.
[4] S. Singh, A. S. Hosen, and B. Yoon, "Blockchain security attacks, challenges, and solutions for the future distributed iot network," *Ieee Access*, vol. 9, pp. 13 938–13 959, 2021.
[5] M. Panahandeh, A. Hamou-Lhadj, M. Hamdaqa, and J. Miller, "Serviceanomaly: An anomaly detection approach in microservices using distributed traces and profiling metrics," *Journal of Systems and Software*, vol. 209, p. 111917, 2024.

[6] L. Meng, F. Ji, Y. Sun, and T. Wang, "Detecting anomalies in microservices with execution trace comparison," *Future Generation Computer Systems*, vol. 116, pp. 291–301, 2021.

[7] T. Li, S. Zhang, Y. Feng, J. Xu, Y. Xie, W. Qiao, and J. Ma, "Logwf: Anomaly detection for distributed systems based on log workflow mining," *The 45th IEEE International Conference on Distributed Computing Systems*, 2025.

[8] B. Vafaie, M. Shamsi, M. S. Javan, and K. El-Khatib, "A new statistical method for anomaly detection in distributed systems," in *2020 IEEE Canadian Conference on Electrical and Computer Engineering (CCECE)*. IEEE, 2020, pp. 1–4.

[9] C. Tu, M. Chen, L. Zhang, L. Zhao, D. Wu, and Z. Yue, "Towards efficient multi-granular anomaly detection in distributed systems," *Array*, vol. 21, p. 100330, 2024.

[10] J. A. Cid-Fuentes, C. Szabo, and K. Falkner, "Adaptive performance anomaly detection in distributed systems using online svms," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 5, pp. 928–941, 2018.

[11] M. Du, F. Li, G. Zheng, and 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*, 2017, pp. 1285–1298.

[12] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, 2019, pp. 807–817.

[13] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[14] R. Vaarandi and 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.

[15] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2021, pp. 492–504.

[16] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 1448–1460.

[17] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Transactions on Dependable and Secure Computing*, vol. 15, no. 6, pp. 931–944, 2017.

[18] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[19] S. Yu, P. He, N. Chen, and Y. Wu, "Brain: Log parsing with bidirectional parallel tree," *IEEE Transactions on Services Computing*, vol. 16, no. 5, pp. 3224–3237, 2023.

[20] M. Du and F. Li, "Spell: Streaming parsing of system event logs," in *2016 IEEE 16th International Conference on Data Mining (ICDM)*. IEEE, 2016, pp. 859–864.

[21] M. Cosler, C. Hahn, D. Mendoza, F. Schmitt, and C. Trippel, "nl2spec: Interactively translating unstructured natural language to temporal logics with large language models," in *International Conference on Computer Aided Verification*. Springer, 2023, pp. 383–396.

[22] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext. zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[23] A. Heußner, T. Le Gall, and G. Sutre, "Mcscm: a general framework for the verification of communicating machines," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 2012, pp. 478–484.

[24] L. T. K. Hien and N. Gillis, "Algorithms for nonnegative matrix factorization with the kullback–leibler divergence," *Journal of Scientific Computing*, vol. 87, no. 3, p. 93, 2021. [Online]. Available: https://doi.org/10.1007/s10915-021-01504-0

[25] C. Lemieux, D. Park, and I. Beschastnikh, "General ltl specification mining (t)," in *2015 30th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2015, pp. 81–92.

[26] C. Cangea, P. Veličković, N. Jovanović, T. Kipf, and P. Liò, "Towards sparse hierarchical graph classifiers," 2018. [Online]. Available: https://arxiv.org/abs/1811.01287

[27] J. Wang, C. Zhao, S. He, Y. Gu, O. Alfarraj, and A. Abugabah, "Loguad: log unsupervised anomaly detection based on word2vec," *Computer Systems Science and Engineering*, vol. 41, no. 3, p. 1207, 2022.

[28] X. Li, P. Chen, L. Jing, Z. He, and 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)*. IEEE, 2020, pp. 92–103.

[29] C. Zhang, X. Wang, H. Zhang, J. Zhang, H. Zhang, C. Liu, and P. Han, "Layerlog: Log sequence anomaly detection based on hierarchical semantics," *Applied Soft Computing*, vol. 132, p. 109860, 2023.

[30] R. Yang, D. Qu, Y. Gao, Y. Qian, and Y. Tang, "Nlsalog: An anomaly detection framework for log sequence in security management," *IEEE Access*, vol. 7, pp. 181 152–181 164, 2019.

[31] X. Ma, Y. Li, J. W. Keung, X. Yu, H. Zou, Z. Yang, F. Sarro, and E. T. Barr, "Practitioners' expectations on log anomaly detection," *ArXiv*, vol. abs/2412.01066, 2024. [Online]. Available: https://api.semanticscholar.org/CorpusID:274436412

[32] T. Li, S. Zhang, Y. Feng, J. Xu, Z. Ma, Y. Shen, and J. Ma, "Heuristic-based parsing system for big data log," in *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*, 2024, pp. 2329–2334.

[33] L. Chen, Q. Dang, M. Chen, B. Sun, C. Du, and Z. Lu, "Berthtlg: Graph-based microservice anomaly detection through sentence-bert enhancement," in *International Conference on Web Information Systems and Applications*. Springer, 2023, pp. 427–439.

[34] Z. He, Y. Tang, K. Zhao, J. Liu, and W. Chen, "Graph-based log anomaly detection via adversarial training," in *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*. Springer, 2023, pp. 55–71.

[35] S. Luftensteiner and P. Praher, "Log file anomaly detection based on process mining graphs," in *International Conference on Database and Expert Systems Applications*. Springer, 2022, pp. 383–391.

[36] H. Guo, Y. Guo, J. Yang, J. Liu, Z. Li, T. Zheng, L. Zheng, W. Hou, and B. Zhang, "Loglg: Weakly supervised log anomaly detection via log-event graph construction," in *International Conference on Database Systems for Advanced Applications*. Springer, 2023, pp. 490–501.

[37] K. Fei, J. Zhou, L. Su, W. Wang, and Y. Chen, "Log2graph: A graph convolution neural network based method for insider threat detection," *Journal of Computer Security*, no. Preprint, pp. 1–24, 2024.

[38] K. Fei, J. Zhou, L. Su, W. Wang, Y. Chen, and F. Zhang, "A graph convolution neural network based method for insider threat detection," in *2022 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Big Data & Cloud Computing, Sustainable Computing & Communications, Social Computing & Networking (ISPA/BDCloud/SocialCom/SustainCom)*. IEEE, 2022, pp. 66–73.

[39] J. Ge, T. Li, and Y. Wu, "Anomaly classification with unknown, imbalanced and few labeled log data," 2023.

[40] B. Zhang, H. Zhang, V.-H. Le, P. Moscato, and A. Zhang, "Semi-supervised and unsupervised anomaly detection by mining numerical workflow relations from system logs," *Automated Software Engineering*, vol. 30, no. 1, p. 4, 2023.

[41] C. Egersdoerfer, D. Zhang, and D. Dai, "Clusterlog: Clustering logs for effective log-based anomaly detection," in *2022 IEEE/ACM 12th Workshop on Fault Tolerance for HPC at eXtreme Scale (FTXS)*. IEEE, 2022, pp. 1–10.

[42] S. Sun and Q. Li, "A behavior change mining method based on complete logs with hidden transitions and their applications in disaster chain risk analysis," *Sustainability*, vol. 15, no. 2, p. 1655, 2023.

[43] W. Meng, F. Zaiter, Y. Zhang, Y. Liu, S. Zhang, S. Tao, Y. Zhu, T. Han, Y. Zhao, E. Wang *et al.*, "Logsummary: Unstructured log summarization for software systems," *IEEE Transactions on Network and Service Management*, vol. 20, no. 3, pp. 3803–3815, 2023.

[44] M. Catillo, A. Pecchia, and U. Villano, "Autolog: Anomaly detection by deep autoencoding of system logs," *Expert Systems with Applications*, vol. 191, p. 116263, 2022.

[45] Y. Feng, J. Xu, and L. Weymouth, "University blockchain research initiative (ubri): Boosting blockchain education and research," *IEEE Potentials*, vol. 41, no. 6, pp. 19–25, 2022.