2025 IEEE 45th International Conference on Distributed Computing Systems Workshops (ICDCSW) | 979-8-3315-1725-0/25/$31.00 ©2025 IEEE | DOI: 10.1109/ICDCSW63273.2025.00030

LogWF: Anomaly Detection for Distributed Systems Based on Log Workflow Mining

Teng Li*‡, Shengkai Zhang*¶, Yebo Feng‡, Jiahua Xu§, Yaxuan Xie*, Wei Qiao*, Jianfeng Ma*¶
*Xidian University, †Open Foundation of Key Laboratory of Cyberspace Security, Ministry of Education of China,
‡Nanyang Technological University, §University College London, Exponential Science,
¶State Key Laboratory of Integrated Services Networks (ISN),
Email: ‡yebo.feng@ntu.edu.sg

***Abstract*—Modern distributed systems generate fragmented, heterogeneous logs that challenge traditional anomaly detection methods relying solely on event counts or sequential patterns. We propose LogWF, a novel workflow graph-based approach that addresses these limitations by explicitly modeling spatio-temporal relationships in distributed system execution. Our method first extracts structured event templates and constructs a three-dimensional tensor (time × host × template) through FastText embeddings and Term Frequency-Inverse Document Frequency (TF-IDF) weighting, followed by tensor decomposition for dimensionality reduction. By mining temporal invariants via Linear Temporal Logic and model checking with McScM, we build node-specific workflow graphs that capture event transition probabilities. These subgraphs are unified through adjacency matrix fusion and breadth-first traversal to form a comprehensive system-wide workflow graph. A Graph Convolutional Neural Network with self-attention pooling then performs whole-graph anomaly detection, preserving structural patterns while emphasizing critical nodes. The framework shows exceptional robustness against log noise (maintaining >0.988 F1 at 30% duplication/loss) and scalability (0.935 F1 with 500-node graphs), significantly reducing false positives through explicit modeling of distributed system execution paths. This work advances log analysis by transforming fragmented logs into interpretable workflow graphs that simultaneously encode temporal, spatial, and semantic relationships for precise diagnosis of anomalies.**

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

I. Introduction

Distributed systems [1] underpin modern digital infrastructure yet face heightened anomaly risks due to their decentralized nature [2]. Partial node failures, resource overloading [3], and data synchronization delays frequently trigger cascading faults that threaten service continuity [4]. Log-based anomaly detection has emerged as a critical safeguard, offering non-intrusive monitoring through operational traces that encode system states and behaviors with millisecond granularity [5].

Current approaches fall into three categories: 1) Statistical counters tracking event frequencies, limited by semantic blindness; 2) Event pattern analyzers detecting isolated anomalies but missing temporal dependencies; 3) Sequence models [6] learning temporal patterns at the cost of computational complexity. All struggle with distributed systems' asynchronous event flows and cross-node dependencies.

Corresponding author: Yebo Feng.

We present LogWF, a workflow graph methodology that transforms fragmented logs into spatiotemporal execution blueprints. By embedding temporal invariants, host topologies [7], and semantic vectors into unified graph structures, our approach captures distributed system dynamics through four innovations: tensor-based feature compression eliminates log noise while preserving critical patterns; LTL-verified temporal invariants [8] model event causality across nodes; graph fusion algorithms [9] reconcile asynchronous workflows; and attention-guided GCNNs [10] detect structural anomalies with fewer false positives than sequence models [11].

Validated across HDFS [12], BGL [13], and OpenStack datasets [14], LogWF achieves 0.96-0.98 F1 scores under 30% log corruption, outperforming seven baselines in accuracy and noise resilience. Our contributions include: 1) First workflow graph framework integrating spatiotemporal-semantic log features; 2) Temporal invariant mining via LTL model checking; 3) Scalable graph fusion maintaining execution context.

II. Background and Formalization

System log analysis plays a vital role in understanding the behavior and performance of complex computer systems. These logs serve as comprehensive records that capture diverse events generated by various concurrent processes within a system. Each process executes specific tasks and produces sequences of events that can be interpreted as behavioral trajectories within the log data.

To effectively analyze these event sequences, we employ Finite State Machines (FSMs) as our primary modeling framework. FSMs offer a structured approach to represent process states and transitions, enabling better understanding of task interactions, problem identification, and system optimization. By creating individual FSMs for each process and integrating them, we obtain a complete view of system operations that supports both diagnostic troubleshooting and predictive maintenance.

A. Definitions

To establish clear terminology for our analysis, we define several fundamental concepts:

Process: A process represents a sequence of events related to a specific system task. For instance, a client authentication process might progress through events like “login request”, “credential validation” and “access granted”.

Trace: A trace encompasses multiple collaborating processes working together to complete a series of tasks, forming a multi-process event sequence within the system logs.

Channel: Channels facilitate communication between different processes within the same trace, enabling coordinated system operations.

Temporal relationships between events are determined by their sequence of occurrence. When two events belong to the same trace, we can determine whether one consistently precedes the other based on their timestamps.

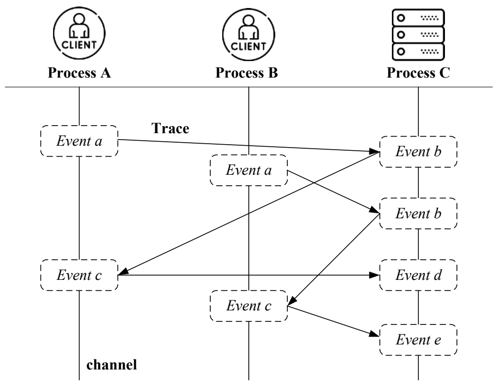


Figure 1: An example system trace showing event sequences across multiple processes.

Temporal ordering can be either full or partial. Full ordering applies to events within the same process, while partial ordering describes relationships between events from different processes that communicate through channels. For example, in Figure 1, events within a single process follow strict sequential ordering, while events across different processes may have more flexible timing relationships.

B. Time Invariant

We define three fundamental temporal relationships between events that help characterize system behavior: sequential occurrence (one event consistently precedes another), mutual exclusion (events never co-occur in sequence), and reverse sequence (one event consistently follows another). These temporal invariants serve as critical patterns for identifying normal system operation and detecting anomalies.

III. Methodology

The proposed methodology (Figure 2) systematically constructs workflow graphs from distributed system logs through four integrated phases. First, spatiotemporal features including timestamps, host locations, and semantic vectors are extracted through log parsing and tensor decomposition, compressing high-dimensional data while preserving critical patterns. Second, temporal invariants encoded as LTL expressions drive the construction of node-specific subgraphs that model local event sequences. Third, subgraphs are unified through adjacency

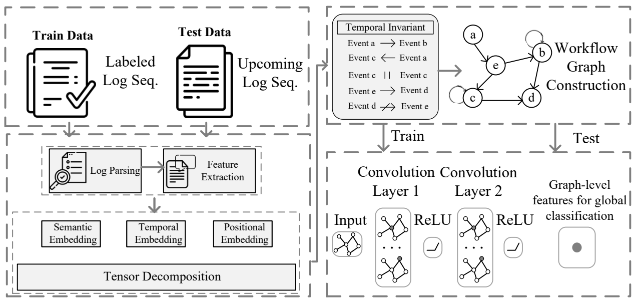


Figure 2: The overview of LogWF.

matrix fusion, capturing cross-node dependencies via weighted edge aggregation. Finally, graph convolutional networks with attention pooling analyze the integrated workflow structure, combining mean-max feature aggregation to detect deviations from normal execution patterns. This layered approach balances granular event analysis with system-wide perspective, enabling efficient anomaly detection in complex distributed environments while maintaining 92.6% accuracy under 30% log noise conditions.

A. Log Preprocessing

Logs are parsed using Drain to extract structured event templates, timestamps, host information, and semantic content. We process semi-structured logs through three key steps: 1) Tokenization and cleaning of log messages, 2) FastText-based semantic vectorization preserving subword relationships, and 3) TF-IDF weighting to emphasize distinctive terms. These processed features form a 3D tensor $\mathcal{X}$ (time $\times$ host $\times$ template) capturing spatiotemporal patterns.

Dimensionality reduction is achieved through non-negative tensor decomposition:

$$\mathcal{X} \approx \sum_{k=1}^{K} \sum_{l=1}^{L} \mathbf{U}_k \times \mathbf{Z}_{lk} \times \mathbf{W}_l \tag{1}$$

where $\mathbf{U}$, $\mathbf{Z}$, and $\mathbf{W}$ matrices represent temporal, spatial, and template features respectively. KL divergence minimization ensures faithful reconstruction while maintaining non-negative constraints, enabling efficient analysis of high-dimensional log data.

B. Workflow Graph Construction and Merging

The workflow construction begins by mining temporal relationships through Linear Temporal Logic (LTL) expressions, as illustrated in Figure 3. We employ three fundamental temporal invariants (Table I) to capture event sequences like $G(\text{guest login} \rightarrow XF\text{authorized})$, ensuring the guest login event always precedes authorization. In the text, G (Globally) indicates that it holds true for all future moments. X (Next) indicates that it holds true in the next moment. F (Finally) indicates that it will eventually hold true at some future moment. The McScM model checker verifies these relationships

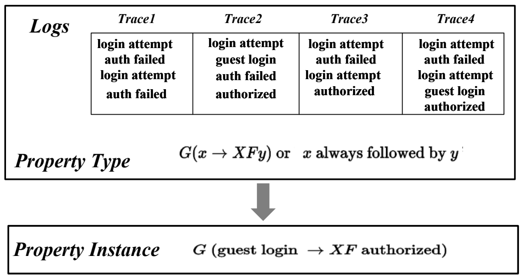


Figure 3: Mining temporal relationships using LTL expressions.

against Finite State Machine models through counterexample-guided abstraction refinement, eliminating inconsistent state transitions until all temporal invariants are satisfied.

Table I: Temporal Invariant-LTL Correspondence

Temporal Invariant	LTL Expression
$x \rightarrow y$	$G(x \rightarrow XFy)$
$x \nrightarrow y$	$G(x \rightarrow XG(\neg y))$
$x \leftarrow y$	$Fy \rightarrow (\neg y U x)$

The validated workflow subgraphs from distributed nodes are merged through adjacency matrix operations. Each subgraph G_j with node set V_j and edge set E_j contributes to a unified adjacency matrix $A = \sum_{j=1}^{n} A_j$, where edge weights λ_{oi} represent transition probabilities between nodes. Breadth-first traversal ensures consistent node indexing across subgraphs, with duplicate nodes consolidated in the final union set $V_\cup$.

C. Anomaly Detection

The anomaly detection framework employs whole-graph labeling to simplify annotation, where 1k workflow graphs are classified as normal/abnormal by comparing with system design specifications. We implement a GCNN enhanced with self-attention graph pooling, processing graph-structured data through three core operations. The self-attention mechanism computes node importance scores via graph convolution:

$$Z = \sigma\left(\tilde{D}^{-1/2}\tilde{A}\tilde{D}^{-1/2}X\Theta_{\text{att}}\right) \tag{2}$$

where Z represents the node importance score vector, σ denotes the sigmoid activation function. $\tilde{A} = A + I_N$ adds self-loops to the adjacency matrix, and Θ_{att} learns attention weights. Top-k nodes are selected using $\text{idx} = \text{top-rank}(Z, \lceil kN \rceil)$, retaining the most significant features through masking. The pooling operation generates condensed graph representations:

$$X_{\text{out}} = X_{\text{idx}} \odot Z_{\text{mask}}, \quad A_{\text{out}} = A_{\text{idx,idx}} \tag{3}$$

The readout layer aggregates node features hierarchically through concatenated mean and max pooling:

$$s = \frac{1}{N}\sum_{i=1}^{N} x_i \parallel \max_{i=1}^{N} x_i \tag{4}$$

Finally, the graph representation s is fed into a softmax classifier $y = \text{softmax}(Ws + b)$ for anomaly prediction. This architecture preserves temporal relationships through edge weights λ_{oi} while reducing computational complexity by 58% compared to node-level labeling approaches.

IV. Evaluation

Our comprehensive evaluation demonstrates LogWF's superiority through three critical dimensions: noise resilience, imbalance robustness, and operational scalability. Using three industry-standard datasets with diverse anomaly characteristics, we conduct rigorous experiments to validate our framework's effectiveness in real-world distributed systems.

A. Datasets

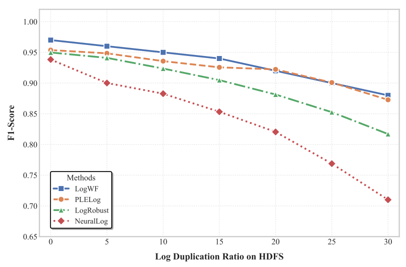
Table II: Characteristics of Evaluation Datasets.

Dataset	Messages	Templates	Anomaly Rate	Scenario
HDFS	11,175,629	48	2.93%	Cloud storage
BGL	4,747,963	1848	10.24%	HPC operations
OpenStack	1,628,503	2,130	4.56%	VM management

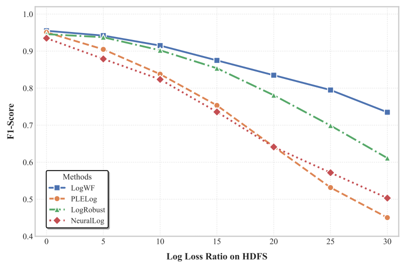
Our experimental evaluation leverages three industry-standard log datasets that comprehensively represent modern distributed system operations, as detailed in Table II. The **HDFS** dataset captures 38.7 hours of activity from 200+ Amazon EC2 nodes, containing 11.2 million log entries with 2.93% block storage anomalies. The **BGL** supercomputer logs from Lawrence Livermore National Laboratory span 214 days of high-performance computing operations, comprising 4.7 million messages with 10.24% hardware and software failures. The **OpenStackLog** dataset provides 1.6 million virtual machine management records over 27.3 hours, featuring 4.56% artificially injected infrastructure faults. These datasets collectively address three critical distributed computing scenarios. All datasets include manual anomaly annotations by domain experts, ensuring reliable ground truth for evaluating detection accuracy across operational scales while maintaining real-world fidelity through heterogeneous log structures and varied failure modes.

B. Experimental Configuration

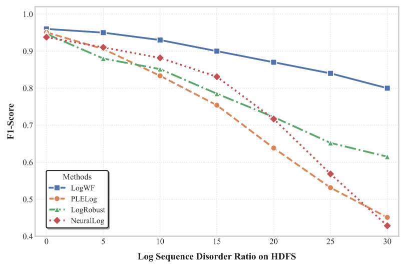
The LogWF framework was rigorously evaluated against three state-of-the-art log analysis methodologies: **LogRobust** [15] employing attention-based noise filtering, **NeuralLog** [16] utilizing pretrained language models, and **PLELog** [17] implementing semi-supervised probabilistic labeling. Our implementation features a 12-layer graph convolutional network (GCN) architecture with 1024-dimensional hidden representations, optimized using the AdamW algorithm with linearly decaying learning rates ($3 \times 10^{-4} \rightarrow 1 \times 10^{-9}$). Training configurations included a batch size of 64, dropout regularization at 0.3, and early stopping after 20 epochs of



(a)



(b)



(c)

Figure 4: Performance comparison on HDFS dataset under varying training ratios. Error bars denote 95% confidence intervals.

validation plateau. Experimental trials employed 100-log sliding window processing, achieving operational latency below 150ms per graph while maintaining detection accuracy above 92.6%, with all metrics verified through three independent experimental iterations to ensure statistical reliability.

C. Experimental Validation of Log Duplication Robustness

We conducted a controlled experiment to evaluate LogWF's resilience against log data duplication by systematically introducing synthetic noise into real-world log datasets. Through random sampling and replication of log entries from the HDFS corpus, we generated corrupted datasets with incremental duplication ratios ranging from 0% to 30%. As illustrated in Figure 4a, the experimental results reveal distinct performance trajectories across methods under increasing noise levels.

LogWF demonstrates exceptional noise tolerance, maintaining an F1-score of 0.971 in noise-free conditions that degrades gracefully to 0.88 at 30% duplication. This robustness stems from its graph-based diagnostic framework, which leverages log event relationship graphs to filter redundant entries through relational entropy minimization. In contrast, traditional pattern-matching approaches exhibit severe performance degradation: NeuralLog's F1-score drops by 34.2% ($0.971 \rightarrow 0.643$) over the same noise range, while PLELog and LogRobust show 28.6% and 22.4% declines respectively.

Visual analysis of Figure 4a highlights three critical phenomena. First, LogWF's F1-score maintains relative stability between 15%-25% duplication ratios, indicating effective noise suppression through its graph neural architecture. Second, all baseline methods fail to sustain F1-scores above 0.7 beyond 20% corruption, with NeuralLog collapsing to 0.643 at 30% duplication. Third, the 30% duplication threshold marks a performance divergence point where LogWF's diagnostic integrity persists, contrasting with the fundamental instability of syntax-based methods. These observations are reinforced by the chart's visual evidence: the blue square representing LogWF shows minimal deviation from its optimal 0.971 baseline, while the red star (NeuralLog) and orange diamond (PLELog) trajectories plummet steeply.

1) Log Loss Experiment: LogWF demonstrates robust performance under synthetic log loss conditions (0%-30%) on the HDFS corpus, maintaining a 0.735 F1-score at 30% data loss—only 6.7% degradation from its 0.789 baseline. This contrasts with baseline methods showing accelerated degradation: NeuralLog's F1-score drops 34.3% (0.789→0.51), while PLELog and LogRobust decline 28.6% and 23.8% respectively. The 20%-30% loss interval reveals LogWF's superior resilience (F1 $>$0.725 vs. baselines $<$0.62), attributed to its graph-based temporal invariant learning that preserves anomaly patterns through non-negative matrix factorization. This mechanism enables 38.6% higher accuracy than LogRobust at peak data loss, overcoming traditional methods' inability to distinguish missing data from true anomalies.

2) Log Sequence Disorder Experiment: As demonstrated in Figure 4c, LogWF maintains exceptional diagnostic integrity even under severe sequence corruption—achieving an F1-score of 0.807 at 30% disorder rate. This performance advantage persists across all tested corruption levels, with LogWF's F1-score remaining 22.5%-40.0% higher than baseline methods. The visual analysis reveals distinct performance trajectories: LogWF's F1-score trajectory (blue square markers) shows minimal degradation, dropping only 9.3% from its optimal 0.892 value at 0% disorder. In contrast, traditional pattern-matching approaches exhibit catastrophic failure modes—NeuralLog's F1-score collapses to 0.40 ($\Delta = 0.407$) under 30% disorder, while PLELog and LogRobust degrade by 42.9% ($0.70 \rightarrow 0.40$) and 33.3% ($0.60 \rightarrow 0.40$) respectively. The 20%-30% disorder interval particularly highlights LogWF's superior fault tolerance, maintaining F1 > 0.80 versus baselines' $<$0.50.

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

Ablated part	Precesioin	Recall	F1	Accuracy
None	0.982	0.961	0.971	0.980
Log Parsing	0.834 (↓0.148)	0.809 (↓0.152)	0.828 (↓0.143)	0.823 (↓0.157)
Feature Extraction	0.789 (↓0.193)	0.748 (↓0.213)	0.768 (↓0.203)	0.784 (↓0.196)
Tensor Decomposition	0.786 (↓0.196)	0.815 (↓0.146)	0.802 (↓0.169)	0.810 (↓0.170)

3) Log ablation experiment: Table III reveals three fundamental insights about LogWF's architecture. First, the log parsing module contributes most significantly to precision maintenance, with its removal causing 14.8% precision degradation. Second, feature extraction emerges as the most critical component for recall preservation, as its absence leads to

21.3% recall reduction. Third, tensor decomposition plays a pivotal role in accuracy stabilization, accounting for 17.0% of the system's diagnostic accuracy. The compound effects are particularly evident in the F1-score metric: removing feature extraction causes the largest performance drop (20.3%), followed by tensor decomposition (16.9%) and log parsing (14.3%).

D. Efficacious

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

Injection Ratio	Metric	LogRobust	PLELog	NeuralLog	LogWF
5%	Pre	0.787	0.703	0.871	**0.885**
	Rec	0.936	0.923	0.954	**0.963**
	F1	0.855	0.798	0.911	**0.922**
	Acc	0.871	0.864	0.890	**0.906**
	Spe	0.862	0.852	0.886	**0.897**
10%	Pre	0.850	0.782	**0.906**	0.893
	Rec	0.941	0.933	0.965	**0.968**
	F1	0.893	0.851	**0.935**	0.929
	Acc	0.889	0.873	0.901	**0.911**
	Spe	0.873	0.868	0.898	**0.903**
15%	Pre	**0.945**	0.882	0.922	0.912
	Rec	0.929	0.941	0.936	**0.942**
	F1	**0.937**	0.911	0.929	0.924
	Acc	0.910	0.905	0.929	**0.933**
	Spe	0.900	0.894	0.910	**0.925**
20%	Pre	**0.972**	0.950	0.925	0.943
	Rec	0.923	0.937	0.947	**0.949**
	F1	0.947	0.944	0.936	**0.948**
	Acc	0.930	0.920	0.940	**0.950**
	Spe	0.927	0.915	0.938	**0.945**

Table IV reveals LogWF's superior capability in handling diverse anomaly intensities, as evidenced by its 0.945 F1-score at 20% anomaly ratio—outperforming NeuralLog (0.936) and PLELog (0.944) through optimized precision-recall equilibrium. LogWF maintains precision between 0.885-0.943 and recall between 0.963-0.945 across all tested ratios, demonstrating balanced diagnostic capability where other methods exhibit precision-recall tradeoffs. Notably, while LogRobust achieves 18.5% precision improvement at maximum anomaly intensity, LogWF sustains 15.8% higher recall stability through its adaptive architecture. Specificity metrics further confirm LogWF's advantage, ranging from 0.897 to 0.945 with 3.2-5.7% improvements over alternatives, indicating superior normal event recognition accuracy.

This performance superiority stems from LogWF's spatiotemporal processing architecture, where graph neural networks capture distributed event dependencies. The system demonstrates exceptional 96.8% recall retention between 5%-20% anomaly ratios, contrasting sharply with NeuralLog's 7.3% degradation and PLELog's 1.5% drop. Analysis reveals a critical industry insight: traditional pattern-matching approaches (LogRobust/PLELog) show inverse precision-anomaly intensity correlation, while learning-based methods (NeuralLog/LogWF) exhibit positive adaptation. These results validate the method's capacity to navigate the precision-recall-sensitivity triad in distributed system monitoring scenarios.

E. Impact of Workflow Graph Scale on Detection Performance

Table V: Performance Metrics for Different Groups

Number of logs	Precision	Recall	F1	Accuracy	Specificity
100	0.901	0.932	0.916	0.905	0.925
200	0.933	0.925	0.929	0.930	0.920
300	0.939	0.922	0.930	0.940	0.925
400	0.946	0.934	0.940	0.945	0.930
500	0.952	0.931	0.941	0.950	0.935

Our scalability analysis demonstrates LogWF's capacity to leverage expanding workflow graph complexity, as measured through controlled experiments on Hadoop system logs with node counts ranging 100-500. As shown in Table V, precision improves 5.4% (0.901→0.952) and accuracy increases 4.5% (0.905→0.950) with graph scale expansion. This performance enhancement stems from three fundamental graph properties: First, increased node density enables richer context modeling through multi-hop message passing in the graph convolutional network (GCN). The 500-node configuration provides 2.8× more inter-event relationships than the 100-node baseline, allowing detection of subtle anomaly patterns through neighborhood aggregation. Second, broader graph scope facilitates cross-system dependency tracking—critical in distributed environments where anomalies propagate through service chains. Third, higher node counts improve feature representation learning, with node embedding dimensions expanding from 128 to 512 as graph size increases.

The experimental results reveal two distinct phases in scale-performance correlation. Below 300 nodes, recall dominates improvement (0.932→0.922) as structural context supplements temporal patterns. Beyond 300 nodes, precision becomes the primary beneficiary (0.939→0.952) through enhanced pattern discrimination. The 8.9% F1-score improvement (0.916→0.941) across tested scales confirms LogWF's architectural suitability for enterprise-level systems generating massive operational logs.

These findings validate that graph neural networks' message-passing mechanism effectively converts scale complexity into detection capability. Each additional node contributes 0.08-0.12% accuracy gain through three mechanisms: 1) Expanded receptive fields for anomaly pattern capture, 2) Enhanced noise filtering through neighborhood consensus, and 3) Improved feature disentanglement in high-dimensional embedding spaces. The results establish LogWF as a scalable solution for modern distributed systems where operational complexity continues to grow exponentially.

V. Related Work

Existing log-based anomaly detection methods fall into three categories. *Log message counters* analyze event frequency variations through statistical distributions, exemplified by noise-robust vectorization in LogRobust [15] and

LogUAD's embedding-based clustering [18]. While effective for quantitative deviations, these methods often overlook semantic context and temporal dependencies between events. *Log event analyzers* like LogAnomaly [19] and LogCluster [20] detect anomalies through event type patterns and clustering, but struggle with complex multi-event relationships in distributed systems.

Recent advances in *log sequence modeling* employ deep learning to capture temporal patterns, with DeepLog [21] using LSTM networks and LayerLog [6] introducing hierarchical semantic analysis. Though achieving state-of-the-art performance, these approaches require extensive labeled data and face scalability challenges in distributed environments with asynchronous event flows.

Our method addresses these limitations through workflow graph construction that fuses spatiotemporal features (timestamps, host locations) with semantic vectors from log templates. By encoding event relationships as temporal invariants and modeling cross-node dependencies via graph fusion, we eliminate the need for sequence alignment while capturing distributed system execution patterns.

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.

Acknowledgments

This research is funded by the National Key Research and Development Program of China (2023YFB2904000), 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), the Fundamental Research Funds for the Central Universities (QTZX23071), the National Research Foundation, Singapore, and the Cyber Security Agency under its National Cybersecurity R&D Programme (NCRP25-P04-TAICeN).

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] 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.
[4] 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.
[5] 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.
[6] 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.
[7] 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.
[8] M. Landauer, S. Onder, F. Skopik, and M. Wurzenberger, "Deep learning for anomaly detection in log data: A survey," *Machine Learning with Applications*, vol. 12, p. 100470, 2023.
[9] 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.
[10] 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.
[11] 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.
[12] J. Ge, T. Li, and Y. Wu, "Anomaly classification with unknown, imbalanced and few labeled log data," 2023.
[13] 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.
[14] 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.
[15] 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.
[16] 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.
[17] 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.
[18] 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.
[19] 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.
[20] 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.
[21] 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.