Current Science
CS 6 (2), 6050-6058 (2026)
Received 10 February 2025 | **Accepted 18 March** 2025 | **Online Available 19 March** 2026
https://doi.org/10.52845/CS/2026-6-2-6

Current Science

Current Science

OPEN ACCESS

ISSN (O) 2795-8639

Original Article

Logdwg: Dynamic Window Graph-Based Log Anomaly Detection with Dual-Modal Graph Convolution

Bo Zhang[1], Mingzhe Li[1], Longxin Yao[1], Xuanran Li[1], Yinling Wang[1,*]

[1]School of Computer Science, Zhengzhou University, 100 Science Avenue, 450001 Zhengzhou, Henan, China

***Corresponding Author: Yinling Wang**

Abstract:
Log anomaly detection is crucial for enhancing the reliability and security of computer systems. Existing methods for mining temporal and relational patterns in logs often exhibit deficiencies. To address these issues, this paper proposes LogDWG, a novel log anomaly detection method based on Dynamic Window Graphs. LogDWG employs a dynamic window segmentation algorithm to adaptively adjust log grouping sizes. It integrates the BERT model to generate semantic embeddings of logs and constructs directed heterogeneous graphs to capture both temporal and relational features between log events. Experimental results demonstrate that LogDWG significantly outperforms traditional methods, such as PCA and SVM, as well as existing deep learning models, such as DeepLog and GLAM. On multiple public datasets, including BGL and HDFS, LogDWG achieves improvements in F1-score of up to 3.6%, validating its effectiveness and accuracy in anomaly detection. The key innovations of this work include the dynamic window grouping strategy, the fusion of log semantics with temporal features, and the design of a dual-modal graph convolutional network.

Keywords: Log anomaly detection, dynamic window grouping, graph neural network, dual-modal graph convolution

Introduction

Logs are ubiquitous in computer systems, and are often presented as semi-structured text generated by logging statements in source code. They provide the primary data source for recording software runtime information. Analyzing log data is a critical technological approach for fault diagnosis, performance optimization, and security monitoring. By mining system operational states, user behaviors, anomalous events, and security threats, log analysis offers data-driven support for compliance auditing, business decisions, and risk management, thereby enhancing system reliability, efficiency, and security.

Log anomaly detection enables timely fault discovery, security enhancement, performance optimization, and compliance assurance through the automated analysis of system logs. It assists operators in quickly identifying potential issues, reduces manual monitoring overhead, improves system stability and user experience, and supports data-driven decision-making, making it an indispensable tool in modern IT management.

As computer systems grow increasingly complex with expanding communication technologies and services, they become more susceptible to various errors and malicious attacks. Consequently, anomaly detection becomes more challenging, often diminishing the effectiveness and efficiency of traditional methods. This underscores the need for deeper research into system log analysis.

Raw logs typically consist of sequences of text lines. For anomaly detection, logs are first grouped and then processed into predefined constant event templates (e.g., file opening activities) and attribute variables (e.g., file names and users). These event templates are arranged in chronological order. However, existing log anomaly detection methods often struggle to simultaneously capture both temporal and relational patterns. Many approaches focus on only one pattern type: some treat logs strictly as temporal sequences, while others primarily model log structures as graphs. This limitation prevents them from fully leveraging the complementary information present in both dimensions, which is often critical for accurate detection. Graph-based approaches have been shown to achieve higher detection accuracy than purely time-series-based methods.

Furthermore, when constructing graphs from logs, conventional approaches group logs based on fixed time intervals or equal entry counts, treating all logs as identical entries. In reality, log sequences processing the same task exhibit higher correlation, whereas logs from different tasks show lower correlation. When a large volume of logs is generated simultaneously, it is more likely they belong to a single task—a phenomenon not adequately addressed in prior research.

To address these problems, we propose LogDWG, a novel log anomaly detection method based on dynamic windows. During the grouping phase, LogDWG groups logs into variably sized segments based on the similarity between log sequences. To better extract syntactic and semantic features, we employ the BERT model [5] for log embedding. Based on the BERT-derived features, we calculate similarity between log groups and perform merging. In the graph representation stage, we use log events as nodes and their sequential order as directed edges to construct a directed graph. Anomalies are then identified through an improved graph neural network. The main contributions of this work are threefold.

1. Dynamic Window Grouping Strategy: By adaptively adjusting window sizes based on event density, it improves the rationality and completeness of log grouping.
2. Fusion of Log Semantics and Temporal Features: Utilizing BERT[5] model and normalizing flow techniques to generate high-quality log embeddings, combined with time decay factors to enhance the robustness of anomaly detection.
3. Dual-Modal Graph Convolutional Neural Network: Through feature fusion of spatial and temporal paths, it effectively enhances the model's ability to identify log anomaly patterns.

Related Work

Log Detection

Log-sequence anomaly detection has recently gained widespread attention. Early studies primarily treated log data as temporal sequences, identifying anomalous sequences by comparing pattern similarities. Subsequent methods improved detection rates by incorporating more log data characteristics. For instance, DeepLog [3] uses both log events and parameters, LogC [6] employs log components and events, and GAE-log [7] utilizes events, components, and log levels for finer-grained detection. Our method focuses on enriching log attributes while reducing the computational overhead of detection algorithms.

Log Parsing

Log parsing aims to separate variables from constants in logs, replacing variables with special tokens to convert raw logs into log templates. LogMine generates templates via cluster grouping. IPLoM [8] uses an iterative partitioning strategy. Ying et al. (2021) proposed a method based on N-gram and frequent pattern mining. According to evaluations by Zhu et al. (2019), Drain generates less noise and achieves higher anomaly detection accuracy compared to other

parsers. Therefore, we employ Drain for log parsing.

Feature Extraction

Feature extraction analyzes semantic and temporal information in templated logs, converting them into vectors learnable by neural networks. Existing methods fall into two categories: index-based and semantic-based. Xu et al. (2009) used event count vectors as input to PCA [1]. LogEvent2vec [9] uses event indices as input to Word2vec [10]. Armand et al. (2016) employed SVM [2] based on event count vectors. These index-based methods lack semantic information, leading to poor model robustness. To address this, LogRobust [11] used FastText, Log Transfer [12] used GloVe [13], and LightLog used Word2vec with the PPA algorithm. Among these, the BERT model [5] considers both contextual environment and semantic information, offering powerful feature extraction capabilities and often outperforming other methods in anomaly detection. Given the semi-structured nature of log data, our approach focuses on both the semantics of entire log statements and the meaning of individual log fields, fusing these features for a more comprehensive representation.

Anomaly Detection

Deep learning-based anomaly detection is a prominent research area. Models like DeepLog [3] employ temporal deep learning but overlook relationships between log entries or attributes. GLAM [4] uses a graph and time series approach, simultaneously considering temporal and relational patterns, which is more conducive to effective log detection.

Design of LogDWG

We propose LogDWG, a log anomaly detection method based on Dynamic Window Graphs. It performs structured modeling and deep analysis of log data through the following core steps: First, raw logs are obtained from the system and dynamically segmented into time windows. Second, log parsing is performed within each window to obtain structured log templates. Third, we employ the BERT model [5] to generate log event embeddings and integrate time interval information into graph node features. Fourth, via an event chain backtracking and merging mechanism, we calculate similarity between windows within a period and merge highly similar ones. Finally, directed graph sequences are constructed using the merged windows as units. The overall construction process is illustrated in Figure 1.

Density-Triggered Adaptive Windowing (DTAW)

Traditional fixed time windows struggle to capture entire burst event processes (e.g., a large volume of logs in a short time), leading to incomplete event capture. To address this, we propose an adaptive segmentation algorithm based on event density. We define a log entry as l, yielding the raw log sequence $L = \{l_1, l_1, \dots, l_n\}$, where Lrepresents the entire log sequence, l_n represents each log entry in the log sequence, and n represents the number of log entries in the log sequence. Setting the initial event window size as , we first calculate the log event arrival rate within Δt:

$$\rho(t) = (\Delta N)/(\Delta t) \quad (1)$$

where ΔN is the number of arriving log entries.When $\rho(t)$ exceeds the threshold ρ_{th} , a dynamic adjustment is triggered. , and the new window size $\Delta t'$ is adjusted according to exponential decay:

$$\Delta t' = \Delta t \cdot e^{-\kappa(\rho(t)-\rho_{th})} \quad (2)$$

where$\kappa$is the decay coefficient. This mechanism allows finer temporal segmentation in high-density log regions while maintaining efficiency in low-density areas.

LogParse

Using HDFS data as an example, each log entry contains LineID, Date, Pid, and Content. We use

the Drain algorithm to extract log templates $l_i^T = \{w_1, w_2, \ldots, w_w\}$, where w_i represents the i-th word in a log sequence of length T. Each template represents a log event, with '<>' denoting parameter values. LogDWG also extracts component information $l_i^C = \{c_1, c_2, \ldots, c_c\}$ and timestamp information $\mathrm{T} = \{\mathrm{t}_1, \mathrm{t}_2, \ldots, \mathrm{t}_\mathrm{t}\}$, Components represent system modules (e.g., software or hardware), indicating log source and location. Timestamps reflect when logs were generated.

Log Embedding

The semantic features of a log event l_n represent the primary content of system activities and are crucial for anomaly detection. For the log sequence L in each time window, LogDWG converts log events into semantic vectors V_i. These vectors must meet two criteria:

1. Reliability: Semantic vectors should effectively represent log events with similar semantics and distinguish log events through semantic differentiation.
2. Completeness: Log semantic vectors should consider information from the entire log event, not just individual words, preserving the significance of different words and their contextual order.

To meet these requirements, LogDWG uses BERT [5] combined with normalizing flow to generate node embeddings. Log events are input into BERT as complete sentences. A [CLS] token is prepended, and the corresponding output is used as the sentence vector:

$$V_i = BERT(CLS, l_i^T) \quad (3)$$

Event Chain Backtracking Merging(ECBN)

In system log analysis, a complete transaction may span multiple time windows, forming event chains with causal relationships. We define an event chain as $C = \{w_1, w_2, \ldots, w_m\}$, where w_i is a time window. A weighted association graph $G = (V, E, \omega)$ is constructed, where node $v_i \in V$ represents a window, and edge weight $\omega(v_i, v_j)$ represents the association strength between nodes, combining semantic similarity and time proximity. The effectiveness of ECBN relies on two components:

1. Semantic Vector Similarity: Calculated via the cosine similarity of window feature vectors:

$$S_{sem} = \frac{V_i \cdot V_j}{||V_i|| \cdot ||V_j||} \quad (4)$$

$$V_i = \frac{1}{|\omega_i|} \sum_{h_k \in \omega_i} h_k \quad (5)$$

2. Time Decay Factor: Relationships between events in adjacent windows are stronger. We use an exponential decay model:

$$S_{time} = e^{-\lambda |t_i - t_j|}, \lambda > 0 \quad (6)$$

The final association strength is obtained through linear combination shown as Eq. (7), where the αandβused to balance the contributions of semantic similarity and temporal proximity to the window association strength.

$$\omega(v_i, v_j) = \alpha S_{sem} + \beta S_{time} \quad (7)$$

When $\omega(v_i, v_j) > \theta$, a merging operation is triggered, using depth-first search for connected component detection to ensure event chain integrity.

Heterogeneous Log Graph Construction

LogDWG merges highly similar time windows to make event chains within each window more complete. The log flow in each merged window is converted into a directed heterogeneous graph $G = (V, E, A)$, where V is the node set (different log events), E is the edge set, and A is the set of node types (components). A type mapping function $\tau(v): V \rightarrow A$ assigns a component to each node, introducing component information to identify log sources. The edge set contains all directed edges $E = (s, t, \omega_{s,t})$, where s is the source node, t is the target node (indicating event $t$occurs after event s), and $\omega_{s,t}$ is the edge weight, representing the frequency of the transition $l_s^T \rightarrow l_t^T$ in the log sequence. These weights capture statistical information about adjacent log events, similar to count vectors, which can improve anomaly detection accuracy (as shown in Figure 2).

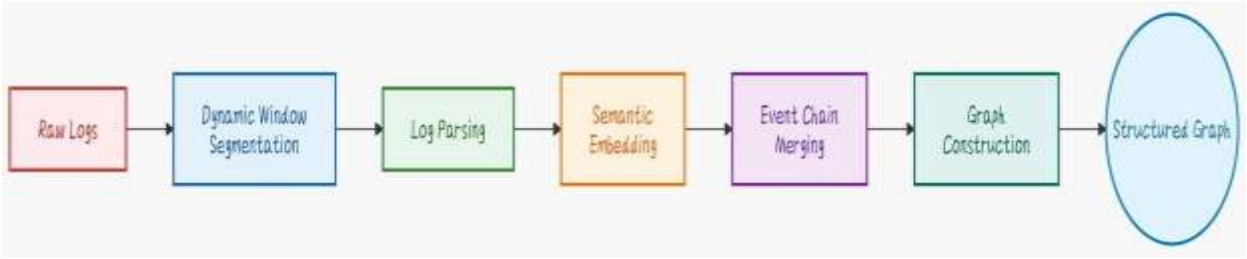

Figure 1 Graph Struction procedure.

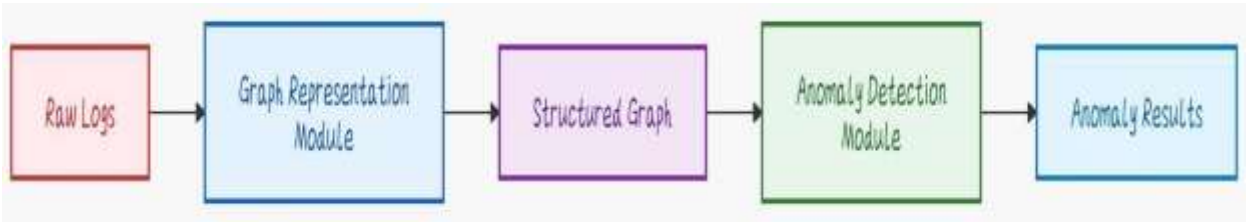

Figure 2 Overall workflow of anomaly detection.

Dual-Modal Anomaly Scoring Head Graph Convolutional Neural Network

We proposea dual-modal graph convolutional neural network for anomaly scoring as shown in Figure 3. The network consists of two core components, introduced in detail as follows.

1. **Time-Aware Directed Graph Convolutional Layer**. This layer employs a dual-path feature fusion architecture to handle both spatial and temporal patterns.

Spatial Path: Uses improved directed graph convolution. The normalized message passing formula is:

$$h_i^{space} = \sigma(\sum_{j\in\mathcal{N}(i)} \frac{e_{ij}}{\sqrt{d_i^+ d_j^-}} W_s h_j) \qquad (8)$$

where: d_i^+ and d_j^- represent the out-degree of node $i$and in-degree of node j , respectively; e_{ij} is the edge weight (determined by log call frequency);$W_s \in \mathbb{R}^{d\times d}$is the learnable parameter matrix.

Time Path: Encodes feature sequences within an event window T using 1D convolution:

$$h_i^{time} = Conv1D(\{x_{i,t}\}_{t=t_0-T}^{t_0}; W_t) \qquad (9)$$

where the convolution kernel $W_t \in \mathbb{R}^{k\times d\times d}$ slides along the temporal dimension k is the convolution kernel size shown as Eq. (10) and Eq. (11).

Adaptive Fusion Gating: Dynamically adjusts path weights via an attention mechanism:

$$g_i = sigmoid(U^{\mathrm{T}}[h_i^{space} || h_i^{time}]) \qquad (20)$$
$$h_i^{out} = g_i \cdot h_i^{space} + (1 - g_i) \cdot h_i^{time} \qquad (31)$$

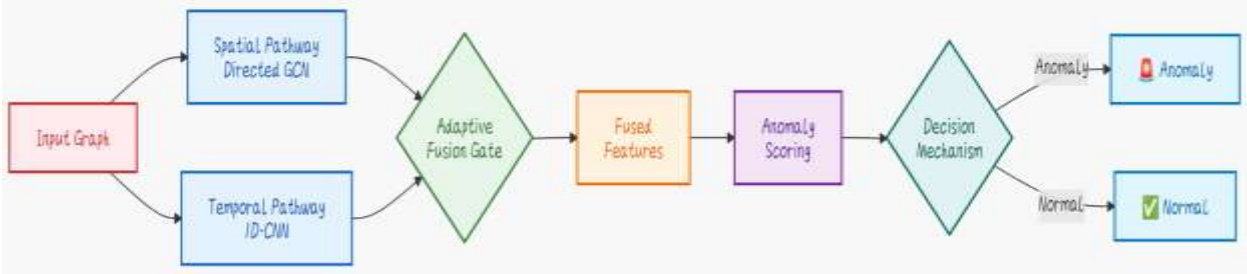

Figure 3 Dual-Modal Anomaly Scoring Head.

2. **Dynamic Log-Weighted Anomaly-Aware Loss Function**. The proposed loss function addresses the limitations of traditional deep SVDD in log anomaly detection: insensitivity to potential anomalies, and susceptibility to extreme values during training.

Loss Function Definition: Given batch embeddings $F \in \mathbb{R}^{N\times d}$F ∈ R(N × d) and anomaly scores $A \in [0,1]^N$, the loss is:

$$L = \frac{1}{N}\sum_{i=1}^{N}[\,1 - \alpha \log(A_i + \epsilon)] \cdot ||\phi(x_i) - c||^2 \quad (42)$$

where $\alpha = 0.3$ is an adjustment factor, $\varepsilon = 10^{-6}$ ensures numerical stability, and $c \in \mathbb{R}^d$ is the SVDD center.

Center Initialization Optimization: The center c is calculated by combining spatial features and anomaly patterns:

$$c = \beta \cdot \frac{1}{N}\sum_{i=1}^{N} \phi\,(x_i) + (1-\beta) \cdot \frac{\sum_{i=1}^{N} A_i \phi(x_i)}{\sum_{i=1}^{N} A_i} \quad (53)$$

where $\frac{1}{N}\sum_{i=1}^{N} \phi\,(x_i)$represents the spatial mean, $\frac{\sum_{i=1}^{N} A_i \phi(x_i)}{\sum_{i=1}^{N} A_i}$represents the anomaly-weighted mean, and β is the balance coefficient.

Experiments

Experimental Setup

Experiments were conducted on an NVIDIA RTX 3080 GPU server using Python 3.10, PyTorch 1.12.1, and CUDA 11.3. For each dataset, 80% was used for training and 20% for testing. We used bert-as-service to obtain 768-dimensional log statement vectors. Hyperparameters were optimized via grid search on validation sets (BGL[14] and HDFS [15]), using F1-score as the metric. Values of hyperparameters are shown in Table 1.

Table 1 Hyperparameters Values.

Hyperparameter	Value	Search Range	Description
λ	0.05	[0.01, 0.1]	Controls the rate of time decay
α	0.7	[0.5, 0.9]	Weight for semantic similarity
β	0.3	[0.1, 0.5]	Weight for temporal similarity
θ	0.85	[0.7, 0.95]	Threshold for window merging
$\mathscr{K}$	0.2	[0.1, 0.5]	Decay coefficient for window adjustment

When association strength exceeds θ, a window merge is automatically triggered. The system merges correlated windows and recursively recalculates similarity with adjacent windows, dynamically expanding the event scope to ensure coherent and complete capture.

Experimental Results and Analysis

Table 2 The Result of HDFS

Methods	Precision	Recall	F1-score
PCA	0.877	0.501	0.638
SVM	0.893	0.611	0.726
DeepLog	0.878	**0.943**	0.909
GLAM	0.928	0.911	**0.919**
LogDWG	**0.938**	0.897	0.917

Table 3 The Result of BGL

Methods	Precision	Recall	F1-score
PCA	0.552	0.638	0.592
SVM	0.756	0.807	0.781
DeepLog	0.874	0.838	0.856
GLAM	0.937	0.923	0.930
LogDWG	**0.961**	**0.932**	**0.946**

LogDWG achieves higher scores than baseline models. PCA performs poorly on complex datasets like BGL as it prioritizes high-frequency features. DeepLog shows significant improvement over SVM, highlighting the advantage of deep learning models. However, DeepLog's metrics are lower than GLAM's due to its use of template indices (losing semantic information) and its inability to model inter-entry relationships, which GLAM addresses via graph structures.

Table 2 shows the performance comparison of different models using the HDFS dataset. As we can see, LogDWG outperforms other schemes in precision, which is due to its dynamic window grouping and dual-modal graph convolution effectively reducing false positives by capturing more complete event contexts and robust spatiotemporal patterns. DeepLog performs best in Recall because its LSTM-based sequence prediction model is highly sensitive to deviations in event sequences, enabling it to detect a wider range of anomalous patterns, albeit at the cost of introducing more false alarms. GLAM achieves a competitive F1-score as it also leverages graph structures to model inter-log relationships, demonstrating the general effectiveness of incorporating relational information. The overall performance of LogDWG on HDFS remains highly competitive, with the precision-centric advantage aligning with its design focus on accurate anomaly identification.

Ablation Experiments

To verify the effectiveness of the two core modules—dynamic window grouping and dual-modal graph convolution—in logDWG, we designed ablation experiments by removing these two modules respectively and comparing the changes in F1-score on the BGL and HDFS datasets. The experimental setup is the same as in Section 4.1, with results shown in the following table:
To verify the effectiveness of the core modules—dynamic window grouping (Win) and dual-modal graph convolution (GCN)—we performed ablation studies (Table 4).

1. Removing dynamic window grouping caused F1-score decreases of 2.6% (HDFS) and 2.5% (BGL), confirming its importance for capturing burst events.
2. Removing dual-modal graph convolution (keeping only the spatial path) caused F1-score decreases of 1.1%~1.4%, demonstrating the value of spatio-temporal feature fusion.
3. Removing both components resulted in the most significant performance degradation (HDFS: -4.1%, BGL: -4.4%), proving their complementary contributions to LogDWG.

Table 4 The Result Ablation Experiments

Methods	Precision	Recall
LogDWG	0.917	0.946
LogDWG-Win	0.891	0.921
logDWG-GCN	0.903	0.932
logDWG-GCN-WIN	0.876	0.902

Conclusion

This paper proposes LogDWG, a dynamic window graph-based method for log anomaly detection, addressing limitations of traditional methods in mining temporal and relational patterns. Through dynamic window segmentation, LogDWG adaptively adjusts log grouping, ensuring fine-grained segmentation in high-density regions while maintaining efficiency in low-density ones. Combined with BERT's semantic embedding capabilities and the construction of directed heterogeneous graphs, LogDWG effectively extracts log features and captures their interrelationships. Experimental results show that LogDWG outperforms baseline methods (PCA, SVM, DeepLog, GLAM) on

public datasets like BGL and HDFS, particularly in precision and F1-score. Ablation experiments verify the crucial roles of dynamic window grouping and dual-modal graph convolution. The former optimizes temporal segmentation, while the latter enhances spatiotemporal modeling; together they improve detection accuracy.

References

1. Xu, W.; Huang, L.; Fox, A.; Patterson, D.A.; Jordan, M.I. Detecting Large-Scale System Problems by Mining Console Logs. In Proceedings of the 27th International Conference on Machine Learning (ICML); Haifa, Israel, 21–24 June 2010; pp. 37–46.
2. Schölkopf, B.; Platt, J.C.; Shawe-Taylor, J.; Smola, A.J.; Williamson, R.C. Estimating the Support of a High-Dimensional Distribution. Neural Computation. 2001;13(7):1443–1471.
3. Du, M.; Li, F.; Zheng, G.; Srikumar, V. Deeplog: Anomaly Detection and Diagnosis from System Logs through Deep Learning. In Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security (CCS); Dallas, TX, USA, 30 October–3 November 2017; pp. 12 85–1298.
4. Zhao, L.; Sawlani, S.; Srinivasan, A.; Akoglu, L. Graph Anomaly Detection with Unsupervised GNNs. In Proceedings of the 2022 IEEE International Conference on Data Mining (ICDM) Short Papers; Orlando, FL, USA, 28 November–1 December 2022; pp. 12 36–1241.
5. Yin, K.; Yan, M.; Xu, L.; Xu, Z.; Li, Z.; Yang, D. Improving Log-Based Anomaly Detection with Component-Aware Analysis. In Proceedings of the 2020 IEEE International Conference on Software Maintenance and Evolution (ICSME); Adelaide, Australia, 28 September–2 October 2020; pp. 746–756.
6. Xie, Y.; Yang, K. Log Anomaly Detection by Adversarial Autoencoders with Graph Feature Fusion. IEEE Transactions on Reliability. 2021;70(3):1226–1238.
7. Makanju, A.A.; Zincir-Heywood, A.N.; Milios, E.E. Clustering Event Logs Using Iterative Partitioning. In Proceedings of the 15th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD); Paris, France, 28 June–1 July 2009; pp. 1255–1264.
8. Wang, J.; Tang, Y.; He, S.; Zhao, C.; Sharma, P.K.; Alfarraj, O. LogEvent2Vec: LogEvent-to-Vector Based Anomaly Detection for Large-Scale Logs in Internet of Things. Sensors. 2020;20(9):2451.
9. Mikolov, T.; Chen, K.; Corrado, G.; Dean, J. Efficient Estimation of Word Representations in Vector Space. In Proceedings of the 1st International Conference on Learning Representations (ICLR); Scottsdale, Arizona, USA, 2–4 May 2013.
10. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y. 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 (ESEC/FSE)*; Tallinn, Estonia, 26–30 August 2019; pp. 807–817.
11. Chen, R.; Zhang, S.; Li, D.; Zhang, Y.; Guo, F.; Meng, W. LogTransfer: Cross-System Log Anomaly Detection for Software Systems with Transfer Learning. In Proceedings of the 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE); Coimbra, Portugal, 12–15 October 2020; pp. 124–135.
12. Pennington, J.; Socher, R.; Manning, C.D. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP); Doha, Qatar, 25–29 October 2014; pp. 1532–1543.

13. Wang, Z.; Tian, J.; Fang, H.; Chen, L.; Qin, J. LightLog: A Lightweight Temporal Convolutional Network for Log Anomaly Detection on the Edge. Computer Networks. 2022;203:108616.
14. W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan. Detecting Large-Scale System Problems by Mining Console Logs. In Proceedings of the ACM Symposium on Operating Systems Principles (SOSP), pages 117–132, 2009.
15. A. J. Oliner and J. Stearley. What Supercomputers Say: A Study of Five System Logs. In Proceedings of the IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), 2007.