

Article

A Mathematical Modeling Framework for Semantic–Temporal Graph Learning in Log Anomaly Detection

Shuo Gao ¹ and Xiaoyan Wang ^{2,*}¹ School of Management Engineering, Capital University of Economics and Business, Beijing 100070, China; 32024210133@cueb.edu.cn² School of System Science and Statistics, Beijing Wuzi University, Beijing 101149, China

* Correspondence: wangxiaoyan@bwu.edu.cn

Abstract

System logs provide valuable information for monitoring system states, identifying failures, and detecting anomalous behaviour in large-scale distributed computing systems. However, their unstructured text, complex semantic dependencies, temporal variation, and severe class imbalance limit conventional methods based on event templates, sequence statistics, or shallow features. To address these challenges, this study proposes a unified mathematical modelling framework for log anomaly detection, instantiated as the Conditional Graph Attention Transformer for Log Analysis (CGATLog). The framework represents temporally ordered log observations as a weighted attributed graph and jointly models semantic representations, event dependencies, temporal associations, and minority-class distributions. A BERT-Transformer encoder first extracts contextual semantic representations from log entries. A weighted log graph is then constructed from event-transition relationships, semantic similarity, and temporal proximity. To alleviate anomaly scarcity, anomalous node representations are generated in the graph embedding space and connected to the original graph through a learnable edge-relation predictor. A semantic–temporal gated graph attention mechanism further regulates the contributions of semantic and temporal associations during neighbourhood aggregation. Experiments on HDFS, BGL, and Thunderbird show that CGATLog achieves competitive log anomaly detection performance. The results also confirm that semantic–temporal graph modelling and anomaly sample augmentation improve robustness, particularly under severe class imbalance.

Keywords: mathematical modelling; security behaviour recognition; log anomaly detection; Transformer; graph neural networks; conditional generative adversarial network

MSC: 68T09



Academic Editors: Marjan Mernik and Jonathan Blackledge

Received: 1 July 2026

Revised: 20 September 2026

Accepted: 1 October 2026

Published: 23 October 2026

Copyright: © 2026 by the authors.

Licensee MDPI, Basel, Switzerland.

This article is an open access article distributed under the terms and

conditions of the [Creative Commons Attribution \(CC BY\)](https://creativecommons.org/licenses/by/4.0/) license.

1. Introduction

Modern computing infrastructures are becoming increasingly complex and heterogeneous with the rapid development of cloud computing, edge computing, distributed storage, high-performance computing, and large-scale networked systems [1]. During operation, servers, storage nodes, operating systems, distributed components, communication devices, and application services continuously generate large volumes of logs [2–5]. These logs record system calls, task execution, resource access, communication states, anomaly alerts, and error recovery. They therefore provide an important basis for system-state monitoring, fault localization, reliability analysis, and log anomaly detection. Compared with

conventional numerical indicators, textual logs contain richer semantic descriptions and behavioural context and can reveal internal execution logic and the evolution of abnormal states. Automatically detecting anomalies in massive heterogeneous logs has consequently become an important problem in system reliability, cybersecurity, and intelligent operations and maintenance.

Early log anomaly detection methods mainly relied on manually defined rules, keyword matching, template statistics, and traditional machine-learning models. These methods typically require expert-designed rules or use log parsers to convert raw logs into event templates before detecting anomalies from event frequencies, sequences, or statistical features [6–8]. Although effective in relatively stable environments, they have clear limitations for large-scale heterogeneous logs. Log text is highly unstructured: the same system behaviour may appear in different textual forms, while different anomalies may share similar templates. Template identifiers or event frequencies alone therefore provide an incomplete representation of log semantics. Modern system logs also exhibit strong contextual and temporal dependencies. A log entry that appears normal in isolation may become anomalous in an unusual context, at an abnormal time interval, or along an unexpected execution path. Effective log anomaly detection must therefore model individual-log semantics together with sequential, temporal, and relational dependencies.

Deep-learning methods have recently been widely applied to log semantic modelling and anomaly detection. Recurrent neural networks, convolutional neural networks, Transformers, and pretrained language models can learn discriminative representations directly from log text or event sequences, reducing dependence on manually designed features. Transformer architectures are particularly effective at modelling long-range dependencies and contextual semantics. Studies based on BERT [9], Transformers, and large language models have achieved promising results on HDFS, BGL, and Thunderbird. However, sequence-oriented models and single-log encoders do not explicitly represent structural relationships among log observations. They may therefore miss important relational information when anomalies arise from event-path deviations, abnormal temporal intervals, or unusual cross-log semantic combinations [10–12].

Graph neural networks offer a natural way to model structural relationships among logs. Log entries, event templates, or system events can be represented as nodes, while execution transitions, semantic associations, and temporal relationships can be represented as edges. Graph neural networks then aggregate information from neighbouring observations and capture relational patterns that conventional sequence models cannot represent explicitly. Graph attention networks are especially suitable because they assign different importance weights to neighbouring nodes. Nevertheless, two limitations remain. First, semantic similarity and temporal relationships are often modelled separately or combined with fixed weights, which cannot capture their varying contributions across log pairs. Second, conventional graph attention usually derives attention from node features and lacks an explicit mechanism for adaptively balancing semantic and temporal associations.

Class imbalance is another major challenge in log anomaly detection. Normal observations usually far outnumber anomalous ones, while anomaly samples are scarce, heterogeneous, and sparsely distributed. Models trained on such data may become biased towards the normal class and show reduced Recall for anomalies. Random oversampling only duplicates existing minority samples and does not increase representation diversity. Generating raw log text may instead introduce semantic inconsistency and template noise. We therefore augment minority samples in the learned graph embedding space, where generated representations can preserve discriminative anomaly characteristics and be integrated directly into graph learning. Because generated nodes have no original event

transitions or timestamps, their structural connections must be estimated carefully to avoid additional topological noise.

To address these challenges, we propose the Conditional Graph Attention Transformer for Log Analysis (CGATLog), a unified mathematical modelling framework for log anomaly detection. CGATLog combines contextual semantic representation, weighted log-graph construction, anomaly sample augmentation, and semantic–temporal graph attention. A BERT-Transformer encoder first extracts contextual semantic features from log entries. Event transitions determine graph connectivity, while semantic similarity and temporal proximity characterize relationships between connected log nodes. Minority anomalous node representations are then generated in the graph embedding space, and a learnable edge-relation predictor estimates their connections to original nodes. Finally, a semantic–temporal gated graph attention mechanism adaptively balances semantic and temporal information during neighbourhood aggregation. Multi-head attention, residual connections, and layer normalization are incorporated to improve representation capacity and training stability.

We evaluate the framework on three public log anomaly detection datasets: HDFS, BGL, and Thunderbird. These datasets originate from distributed file-system and high-performance-computing environments and differ in scale, event structure, temporal characteristics, and anomaly distribution. Accordingly, this study focuses on general system-log anomaly detection rather than on a specific IoT device, protocol, or sensor network. Applying CGATLog to IoT and edge-system logs is a potential extension that requires validation on domain-specific datasets.

The main contributions of this work are summarized as follows:

- We formulate log anomaly detection as a node-level discriminative learning problem over a weighted attributed graph $G = (V, E, X, A, W)$ which jointly represents contextual log semantics, event-transition relationships, and temporal associations. A BERT-Transformer encoder is employed to construct semantic node representations, while graph structures explicitly describe dependencies among log observations.
- We introduce an anomaly sample augmentation mechanism in the graph embedding space to mitigate severe class imbalance. Instead of directly generating raw log text, minority anomalous representations are generated in the latent feature space, and a learnable bilinear edge-relation predictor estimates their structural connections with existing nodes, allowing the generated samples to participate in subsequent graph learning.
- We design a semantic–temporal gated graph attention mechanism that adaptively integrates semantic and temporal association information for different neighbouring node pairs. The learnable gating strategy enables the model to dynamically regulate heterogeneous relational contributions rather than relying on fixed semantic–temporal fusion.
- We evaluate CGATLog on the public HDFS, BGL, and Thunderbird datasets through overall performance comparison, ablation analysis, and class imbalance experiments. The experimental results verify the effectiveness of contextual semantic representation, graph-based relational modelling, anomaly sample augmentation, and semantic–temporal fusion across heterogeneous system-log environments.

2. Literature Review

Deep learning and natural language processing have become important approaches to log anomaly detection. Traditional methods usually parse raw logs into event templates and then detect anomalies from template sequences. Parsing can reduce vocabulary expansion and out-of-vocabulary problems, but it may also discard informative parameters, paths,

and identifiers. Recent studies have therefore explored semantic modelling, graph structure learning, large language models, and class imbalance handling.

For Transformer- and pretrained language model-based semantic modelling, Guo et al. [13] proposed LogFormer, a pretraining and adapter-tuning framework for log anomaly detection. Its Log-Attention mechanism integrates template semantics with parameter information, reducing semantic loss from parsing and improving cross-domain performance on HDFS, BGL, and Thunderbird. Almodovar et al. [14] proposed LogFiT, which self-supervisedly fine-tunes pretrained BERT-like models on raw logs without templates or anomaly labels. By learning normal patterns through masked sentence prediction, LogFiT outperforms DeepLog and LogBERT on the same datasets. Huang et al. [15] proposed LogCTBL, a semi-supervised model combining CNN, TCN, and Bi-LSTM. It uses Drain3 for parsing, BERT for semantic-vector extraction, and HDBSCAN for pseudo-labelling, thereby modelling local features, long-term temporal dependencies, and bidirectional context. These studies demonstrate the value of pretrained language models for log semantics, but most still represent logs primarily as one-dimensional sequences and do not explicitly capture structured event relationships.

Graph neural networks provide a complementary route for modelling complex log topology. Li et al. [16] proposed Logs2Graphs, which represents log events as attributed, directed, weighted graphs and uses OCDiGCN for unsupervised graph-level anomaly detection. Wang et al. [17] proposed LogGT, a heterogeneous-graph transfer-learning method that jointly models log events, components, and time intervals using BERT, Normalizing Flow, and a time-aware Graph Transformer. In network-security settings, Yilmaz and Das [18] combined GCN and GAT to detect anomalous attacks in campus firewall logs. Zhang et al. [19] proposed MTG-GAN for cross-domain log anomaly detection by integrating temporal event graphs, graph contrastive learning, adaptive masking, and GAN-based generation. These graph-based studies improve structural modelling, but the adaptive integration of semantic and temporal relations remains insufficiently explored.

Large language models have also introduced new approaches to log anomaly detection. Pan et al. [20] proposed RAGLog, which retrieves similar normal logs and uses a large language model for zero-shot anomaly judgement. Duan et al. [21] extended retrieval-augmented generation with active learning in EagerLog, reducing annotation requirements by dynamically expanding a high-quality knowledge base. Zhang et al. [22] proposed XRAGLog to reduce prompt-token consumption through hierarchical compression, while He et al. [23] used large language models to enrich log-event semantics in LLMeLog. Yang et al. [24] proposed LogLLaMA, which combines LLaMA2 fine-tuning with reinforcement learning and Top-K selection. Other studies have addressed weak supervision [25], variable-level anomaly detection [26], cross-system generalization [27], insider-threat detection [28], and GAN-based reconstruction [29]. Although these methods broaden the modelling capabilities of log anomaly detection, large-model approaches may incur substantial inference and deployment costs.

Overall, existing log anomaly detection methods have advanced semantic modelling, graph representation learning, generative augmentation, and large language model-based analysis [30–32]. However, sequence and language models still have difficulty capturing nonlinear structural relationships among log events. Graph-learning methods improve relational modelling, but semantic and temporal associations are often integrated with fixed or weakly adaptive strategies. Class imbalance also remains insufficiently addressed, and generative methods rarely explain how synthetic anomalous representations should be connected to the original log graph. CGATLog addresses these gaps through semantic-temporal relation modelling, graph embedding anomaly augmentation, learnable topology reconstruction for generated nodes, and graph attention-based anomaly classification.

3. Mathematical Formulation of the Proposed CGATLog Framework

3.1. Overall Framework

Let $\mathcal{O} = (o_1, o_2, \dots, o_N)$ denote a temporally ordered sequence of log observations, where each observation $o_i = (m_i, t_i)$ contains semantic content m_i and a timestamp t_i . Security behaviour recognition is formulated as learning a mapping $\phi_{\mathcal{O}}: \mathcal{O} \rightarrow [0,1]^N$, where $\phi_{\mathcal{O}}(\mathcal{O})_i$ denotes the predicted probability that observation o_i corresponds to anomalous behaviour. To implement this mapping, CGATLog transforms the observation sequence into a weighted attributed graph $G = (V, E, X, A, W)$ and performs node-level classification over the constructed graph. The following subsections provide the mathematical formulations of graph construction, conditional anomaly node generation and semantic-temporal graph attention.

CGATLog addresses three characteristics of large-scale system logs: complex semantic relationships, strong temporal dependencies, and scarce anomaly samples. As shown in Figure 1, raw log sequences are transformed into a weighted log graph that jointly represents semantic features, event-sequence relationships, and temporal information. A conditional generative adversarial network then augments anomalous node embeddings in the feature space, and an improved graph attention network performs binary log anomaly classification. The framework contains three modules: hybrid log-graph construction, conditional graph-balancing generation, and improved graph attention anomaly detection. The first module preprocesses raw logs, extracts BERT-Transformer representations, and constructs the weighted graph from event templates and timestamps. The second generates minority-class anomaly embeddings and predicts their graph connections. The third adaptively fuses semantic and temporal associations and uses residual connections and layer normalization to improve training stability.

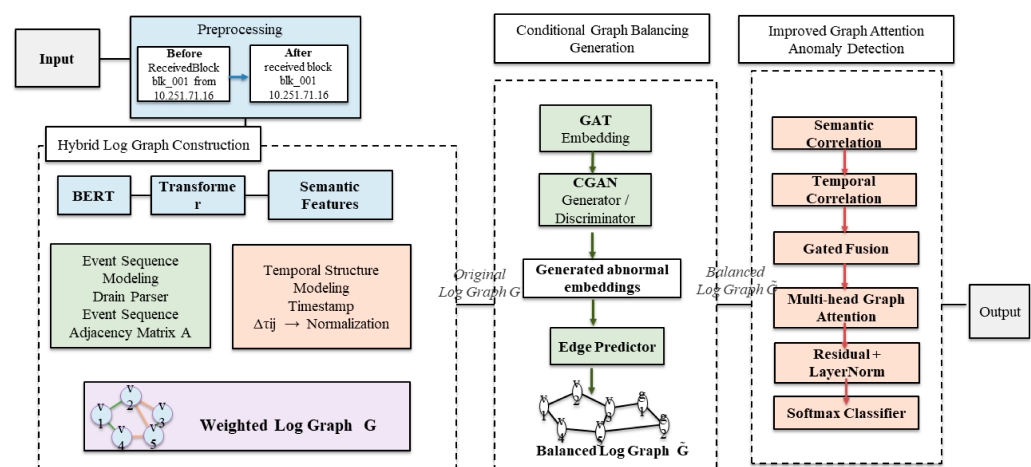


Figure 1. Overall architecture of the proposed CGATLog model.

3.2. Hybrid Log-Graph Construction Module

The hybrid log-graph construction module converts raw log sequences into graph structures containing semantic, sequential and temporal information. Let the raw log sequence be denoted as Equation (1):

$$L = \{l_1, l_2, \dots, l_N\} \quad (1)$$

where N is the number of logs and l_i denotes the i -th log entry. This module first preprocesses the log text, then extracts semantic features, parses log-event templates, computes temporal intervals and finally constructs a weighted log graph as shown in Equation (2):

$$G = (V, E, X, A, W) \quad (2)$$

where V is the node set, E is the edge set, X is the node-feature matrix, A is the adjacency matrix and W is the edge-weight matrix.

3.2.1. Log Data Preprocessing

Raw system logs usually contain many symbols, delimiters and formatting elements that are not directly informative for anomaly detection. Therefore, preprocessing is required before semantic modelling. We first split compound words according to camel-case naming rules and convert all letters to lowercase. For example, the log field `DataNode` is split into `data` and `node`. Regular expressions are then used to remove stop words, punctuation marks and special delimiters, reducing the interference of redundant text in semantic modelling. Unlike conventional text preprocessing, the proposed method preserves numerical variables in logs, such as IP addresses, port numbers, block identifiers, block sizes and time-related parameters. These variables are often closely related to system operating states and may indicate abnormal resource access, communication failures or execution-state changes during anomaly detection. Thus, the preprocessed logs remove irrelevant noise while preserving key information associated with system behaviour. The preprocessed log sequence is denoted as Equation (3):

$$L' = \{l'_1, l'_2, \dots, l'_N\} \quad (3)$$

where l'_i represents the cleaned textual form of the i -th log entry.

3.2.2. BERT-Transformer Semantic Feature Extraction

To obtain contextual semantic representations of logs, we use Transformer encoder [33] and BERT [34] for semantic encoding. BERT extracts the internal contextual semantics of each individual log entry, whereas an additional Transformer encoder models sequence-level dependencies among multiple logs. For the i -th preprocessed log l'_i , special tokens [CLS] and [SEP] are first added to the beginning and end of the log, respectively, and the resulting sequence is fed into the BERT encoder. Let the token representations produced by BERT be

$$U_i = \{u_{i,1}, u_{i,2}, \dots, u_{i,m_i}\} \quad (4)$$

where m_i is the number of tokens in the i -th log and $u_{i,r}$ denotes the contextual representation of the r -th token. To obtain an overall semantic vector for the log entry, mean pooling is applied:

$$s_i = \frac{1}{m_i} \sum_{r=1}^{m_i} u_{i,r} \quad (5)$$

where s_i denotes the initial semantic vector of the i -th log. Relying only on the semantic representation of an individual log is insufficient for capturing contextual relationships within log sequences. For example, a log entry may be normal along a valid execution path but anomalous when it appears in a specific preceding and succeeding context. Therefore, the initial semantic vectors of all logs are assembled into a sequence matrix:

$$S = [s_1, s_2, \dots, s_N]^T \quad (6)$$

The matrix S is then fed into an additional Transformer encoder to learn contextual dependencies among log entries:

$$H = \text{TransEnc}(S) \quad (7)$$

where $\text{TransEnc}(\cdot)$ denotes the Transformer encoder, $H \in \mathbb{R}^{N \times d}$ is the enhanced log semantic feature matrix and d is the feature dimension. The semantic feature of the i -th node is defined as $h_i = H_{i,:}$. The final node-feature matrix is therefore $X = H$.

This design enables each log node to contain not only its own semantic information, but also contextual behavioural patterns within the log sequence.

3.2.3. Log-Event Sequence Modelling

Log-event sequences reflect behavioural paths during system execution. Normal system operation usually follows relatively stable event-transition patterns, whereas anomalous states may lead to event-order deviations, missing key steps or sudden increases in abnormal events. We therefore use log-event sequences to determine edge connections in the log graph. The Drain parser [7] is used to map raw logs to event templates. Let the event-template mapping function be

$$\phi(\cdot): l_i \rightarrow e_i \quad (8)$$

where e_i denotes the event template corresponding to the i -th log entry. The log-event sequence can then be represented as $E_{\text{seq}} = \{e_1, e_2, \dots, e_N\}$.

To capture local execution-context relationships beyond immediate adjacency, we adopt a sliding window strategy for edge construction. Given a window size w , a directed edge from node v_i to node v_j is created when v_j occurs within the following w positions of v_i in the event sequence. Formally, the adjacency relationship is defined as

$$A_{ij} = \begin{cases} 1, & 1 \leq j - i \leq w, \\ 0, & \text{otherwise.} \end{cases} \quad (9)$$

Under this definition, each node v_i can have at most w successors and w predecessors, and its neighbourhood N_i therefore contains at most $2w$ nodes. In all experiments, the window size is set to $w = 5$. When multiple instances of the same event template occur within the window, each instance is retained as an independent node and forms an independent edge with its own edge weight computed according to Equations (11)–(13). The resulting edge set captures local execution-context information among log events within a bounded temporal range.

3.2.4. Temporal Structure Feature Modelling

System logs are generated in temporal order, and the time intervals between logs reflect the operating rhythm of the system. Anomalies are often accompanied by response delays, bursts of logs within a short period or abnormal increases in event intervals. Therefore, temporal-structure information is incorporated into the edge-weight modelling process of the log graph. Let the timestamp of the i -th log be τ_i . The temporal interval between nodes v_i and v_j is defined as $\Delta\tau_{ij} = |\tau_j - \tau_i|$.

Because the numerical scales of time intervals may vary substantially across datasets, min–max normalization is applied:

$$\widehat{\Delta\tau}_{ij} = \frac{\Delta\tau_{ij} - \Delta\tau_{\min}}{\Delta\tau_{\max} - \Delta\tau_{\min} + \varepsilon} \quad (10)$$

where $\Delta\tau_{\min}$ and $\Delta\tau_{\max}$ denote the minimum and maximum temporal intervals among all candidate edges, respectively, and ε is a small constant used to avoid division by zero. The normalized temporal interval satisfies $\widehat{\Delta\tau}_{ij} \in [0, 1]$.

3.2.5. Weighted Log-Graph Construction

After obtaining node semantic features, event-sequence relationships and temporal-structure information, we construct a weighted log graph. For nodes v_i and v_j , the cosine similarity between their semantic vectors is first computed as

$$\rho_{ij} = \frac{h_i^\top h_j}{\|h_i\|_2 \|h_j\|_2 + \varepsilon}, \quad (11)$$

where ρ_{ij} denotes the semantic relevance between nodes v_i and v_j . The semantic similarity and normalized temporal interval are then combined to obtain the edge weight:

$$\omega_{ij} = \beta \rho_{ij} + (1 - \beta) (1 - \widehat{\Delta\tau}_{ij}) \quad (12)$$

where $\beta \in [0,1]$ is a weighting coefficient that controls the relative contributions of semantic relevance and temporal proximity. The term $1 - \widehat{\Delta\tau}_{ij}$ represents temporal closeness, meaning that two logs with a shorter time interval have a stronger temporal association. The edge-weight matrix is finally defined as

$$W_{ij} = \begin{cases} \omega_{ij}, & A_{ij} = 1, \\ 0, & A_{ij} = 0. \end{cases} \quad (13)$$

Thus, the log graph is represented as $G = (V, E, X, A, W)$.

In this graph, each node corresponds to one log entry, node features contain semantic contextual information extracted by BERT-Transformer, edge connections reflect event-execution paths, and edge weights jointly encode semantic relevance and temporal-structure information.

3.3. Conditional Graph-Balancing Generation Module

In system log datasets such as BGL, Thunderbird and HDFS, anomalous logs are usually far fewer than normal logs. Such class imbalance can bias the training process towards the majority class and reduce the recall of anomalous logs. To alleviate this problem, we design a conditional graph-balancing generation module, which uses a conditional generative adversarial network to generate minority-class anomaly node embeddings in the graph embedding space. It should be emphasized that this module does not generate raw log text or new log templates. Instead, it generates anomalous node representations in the embedding space after graph feature extraction. This avoids the semantic uncontrollability introduced by text-level generation and allows the generated samples to participate directly in subsequent graph neural network training.

3.3.1. Graph Embedding Feature Extraction

To enable the generative model to learn the distribution of nodes within the graph structure, we first use a graph attention layer based on GAT [35] to transform the original node features into graph embeddings. For node v_i , its neighbourhood is denoted as N_i . According to the sliding window adjacency defined in Equation (9), N_i contains at most $2w$ neighbouring nodes, including up to w predecessors and w successors. The unnormalized attention score between node v_i and its neighbouring node v_j is defined as

$$a_{ij} = \text{LeakyReLU}(q^\top [Ph_i \| Ph_j]), \quad (14)$$

where P is a learnable linear transformation matrix, q is an attention vector and $\parallel$ denotes vector concatenation. The attention scores over neighbouring nodes are then normalized using the softmax function :

$$\alpha_{ij} = \frac{\exp(a_{ij})}{\sum_{v_k \in N_i} \exp(a_{ik})}. \quad (15)$$

The graph embedding of node v_i is obtained as

$$z_i = \sigma \left(\sum_{v_k \in N_i} \alpha_{ik} P h_k \right) \quad (16)$$

where $\sigma(\cdot)$ denotes a nonlinear activation function. The embeddings of all nodes form the matrix $Z = [z_1, z_2, \dots, z_N]^T$.

Compared with the original semantic features, Z incorporates both node-level semantic information and local graph structure information, making it more suitable as the learning space for the conditional generative adversarial network.

3.3.2. Conditional Anomaly Node Generation

The conditional generative adversarial network consists of a generator G_θ and a discriminator D_ψ [35–37]. The generator is used to generate anomalous node embeddings, whereas the discriminator distinguishes whether an input embedding comes from a real anomalous node or a generated anomalous node. Let the random noise vector be $r \sim \mathcal{N}(0, I)$, and let c denote the class condition vector. Because this study focuses on binary log anomaly detection, the anomaly class condition is denoted as $c = c_{\text{abn}}$. The anomalous node embedding produced by the generator is shown in Equation (17).

$$\tilde{z}_i = G_\theta(r_i, c_{\text{abn}}) \quad (17)$$

where $\tilde{z}_i$ denotes the generated anomalous node embedding. The discriminator receives either a real or generated node embedding together with the class condition, and outputs the probability that the input is a real sample: $D_\psi(z_i, c_{\text{abn}}) \in [0, 1]$. The adversarial objective of the conditional GAN is defined as

$$L_{\text{cgan}} = \mathbb{E}_{z \sim p_{\text{data}}} [\log D_\psi(z, c_{\text{abn}})] + \mathbb{E}_{r \sim p_r} [\log(1 - D_\psi(G_\theta(r, c_{\text{abn}}), c_{\text{abn}}))] \quad (18)$$

The generator and discriminator are optimized through the following minimax game: $\min_{G_\theta} \max_{D_\psi} L_{\text{cgan}}$. After training, M anomalous node embeddings are generated according to the gap between the numbers of anomalous and normal samples: $\tilde{Z} = \{\tilde{z}_1, \tilde{z}_2, \dots, \tilde{z}_M\}$, where M can be determined by the oversampling ratio or the class balancing target.

In the present binary anomaly detection setting, only the anomalous class is augmented; therefore, the condition vector is fixed to c_{abn} during generation. Consequently, the conditional formulation should not be interpreted as providing multi-class controllable generation. Rather, it explicitly restricts the generator to the minority anomalous distribution and keeps the formulation extensible to future multi-class anomaly settings. In the current binary case, its practical behaviour is close to a minority-class GAN oversampler.

3.3.3. Edge-Relation Prediction for Generated Nodes

Because the generated anomalous node embeddings do not correspond to real log text, they have no original event templates or timestamps and therefore cannot be connected directly according to event-sequence rules. To integrate generated nodes into the original log graph, we design an edge-relation prediction mechanism that estimates the connection probability between generated nodes and existing nodes based on node embeddings. For a generated node $\tilde{v}_i$ and an original node v_j , the connection probability is defined as

$$\tilde{a}_{ij} = \text{sigmoid}(\tilde{z}_i^\top R z_j) \quad (19)$$

where R is a learnable bilinear interaction matrix, and $\tilde{a}_{ij} \in [0, 1]$ denotes the probability that an edge exists between the generated node and the original node. The edge-relation predictor is trained on real nodes and their adjacency matrix. Let $\hat{A}$ denote the predicted adjacency matrix and A denote the ground truth adjacency matrix. The edge-prediction loss is defined as

$$\mathcal{L}_{\text{edge}} = -\frac{1}{|\mathcal{E}_c|} \sum_{(i,j) \in \mathcal{E}_c} [A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log (1 - \hat{A}_{ij})] \quad (20)$$

where $\mathcal{E}_c$ denotes the sparse candidate-edge set, $A_{ij} \in \{0, 1\}$ is the ground truth adjacency label, and $\hat{A}_{ij} \in [0, 1]$ is the predicted connection probability. The edge predictor and its loss are computed only over candidate node pairs within the local sliding window rather than over all possible node pairs, avoiding a dense implementation. The generated node embeddings are concatenated with the original node embeddings to obtain the balanced node embedding matrix:

$$Z^{\text{bal}} = [Z \parallel \tilde{Z}] \quad (21)$$

The adjacency matrix and edge-weight matrix are then updated according to the edge-relation prediction results, yielding the balanced log graph:

$$\tilde{G} = (\tilde{V}, \tilde{E}, Z^{\text{bal}}, \tilde{A}, \tilde{W}) \quad (22)$$

This graph is subsequently fed into the improved graph attention anomaly detection module for binary classification.

3.4. Improved Graph Attention Anomaly Detection Module

After log-graph construction and anomalous-node generation, we design an improved graph attention anomaly detection module to perform node-level anomaly classification on the balanced log graph. As shown in Figure 2, conventional graph attention networks can adaptively aggregate features according to the importance of neighbouring nodes. However, in log anomaly detection, node relationships are determined not only by semantic similarity but also by temporal structure. Relying only on fixed edge weights or a single attention mechanism may therefore be insufficient for characterizing the dynamic semantic and temporal contributions between different log pairs. To address this limitation, we introduce a semantic-temporal gated fusion mechanism, enabling the model to adaptively adjust the weights of semantic association and temporal association for different node pairs. In addition, residual connections and layer normalization are added after each graph attention layer to improve the training stability of deep graph networks.

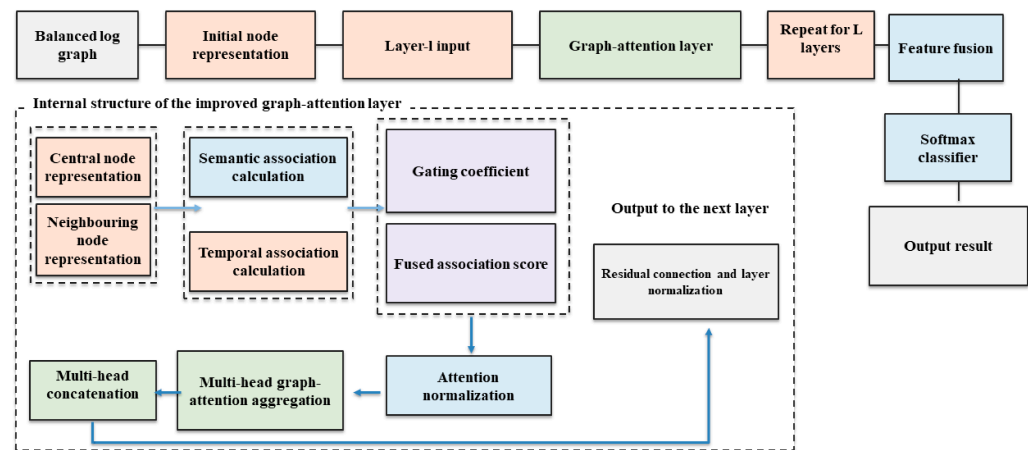


Figure 2. Structure of the improved graph attention anomaly detection module.

3.4.1. Semantic and Temporal Association Calculation

Let the input representation of node v_i at the l -th layer be $g_i^{(l)}$. The initial input is $g_i^{(0)} = z_i^{\text{bal}}$. For node v_i and its neighbouring node v_j , the semantic association score is first computed as

$$s_{ij}^{(l)} = \left(M_s^{(l)} g_i^{(l)} \right)^\top \left(M_s^{(l)} g_j^{(l)} \right) \quad (23)$$

where $M_s^{(l)}$ is the semantic transformation matrix at the l -th layer. For temporal association, the normalized temporal interval is converted into temporal proximity $1 - \hat{\Delta}_{t_{ij}}$, so that temporally closer nodes receive stronger temporal association.

$$q_{ij}^{(l)} = m_t^{(l)} \hat{\Delta}_{t_{ij}} \quad (24)$$

where $m_t^{(l)}$ is a learnable temporal transformation parameter. If an edge involves a generated node, no real timestamp is available. In this case, the mean normalized temporal interval calculated from real edges in the training graph is used as a fixed temporal surrogate. This statistic is computed exclusively from the training set and is applied uniformly to generated-node edges, thereby avoiding the introduction of validation- or test-set information and reducing potential bias in the temporal branch.

3.4.2. Semantic–Temporal Gated Fusion Mechanism

The importance of semantic and temporal information differs across log-node pairs. For example, when two logs are semantically similar but their temporal interval is anomalously large, temporal information may be more informative for anomaly discrimination. Conversely, when the temporal interval is normal but the semantic combination is abnormal, semantic information may become more critical. We therefore design a gated fusion mechanism to adaptively control the contribution of the two types of information. The gating coefficient is defined as

$$\lambda_{ij}^{(l)} = \text{sigmoid} \left(w_g^{(l)} \left[s_{ij}^{(l)} \parallel q_{ij}^{(l)} \right] + b_g^{(l)} \right) \quad (25)$$

where $w_g^{(l)}$ and $b_g^{(l)}$ are learnable parameters, and $\lambda_{ij}^{(l)} \in [0, 1]$. A value of $\lambda_{ij}^{(l)}$ closer to 1 indicates that semantic association is more important for the current node pair, whereas a value closer to 0 indicates that temporal association contributes more strongly. The fused node-association score is computed as

$$e_{ij}^{(l)} = \lambda_{ij}^{(l)} s_{ij}^{(l)} + (1 - \lambda_{ij}^{(l)}) q_{ij}^{(l)} \quad (26)$$

The attention weights over all neighbouring nodes of v_i are then obtained by softmax normalization:

$$\alpha_{ij}^{(l)} = \frac{\exp(e_{ij}^{(l)})}{\sum_{v_k \in N_i} \exp(e_{ik}^{(l)})} \quad (27)$$

Before gated fusion, the semantic association score is normalized using a sigmoid function so that both semantic and temporal association scores lie in $[0,1]$. The fused score is then directly normalized by softmax without an additional tanh operation, avoiding unnecessary compression of the attention logits.

The predefined edge weight W_{ij} is further incorporated into the fused association score before attention normalization. Therefore, W_{ij} acts as a structural prior derived from semantic similarity and temporal proximity, while the gated mechanism performs adaptive pair-specific fusion.

3.4.3. Multi-Head Graph Attention Aggregation

To enhance model expressiveness, we adopt a multi-head attention mechanism. Suppose the l -th layer contains K attention heads. The output of the k -th attention head is

$$o_{i,k}^{(l)} = \sigma \left(\sum_{v_k \in N_i} \alpha_{ij,k}^{(l)} M_k^{(l)} g_j^{(l)} \right) \quad (28)$$

where $M_k^{(l)}$ is the linear transformation matrix of the k -th attention head, and $\alpha_{ij,k}^{(l)}$ is the corresponding attention weight. The multi-head outputs are concatenated as $o_i^{(l)} = \parallel_{k=1}^K o_{i,k}^{(l)}$.

3.4.4. Residual Connection and Layer Normalization

As the number of graph neural network layers increases, node features may suffer from oversmoothing, gradient decay or information degradation during propagation. To mitigate these problems, we adopt the residual connection and layer-normalization mechanisms used in Transformer architectures to stabilize the output of each layer:

$$g_i^{(l+1)} = \text{LayerNorm} \left(g_i^{(l)} + o_i^{(l)} \right) \quad (29)$$

The residual connection preserves node features from the previous layer and prevents important information from being lost during multi-layer propagation, whereas layer normalization stabilizes feature distributions across layers and improves training convergence. After L layers of the improved graph attention module, the final node representation is obtained as $g_i^{(L)}$.

To combine shallow semantic information with deep structural information, the initial node representation and the final node representation are concatenated:

$$r_i = \left[g_i^{(0)} \parallel g_i^{(L)} \right] \quad (30)$$

where r_i is the final node representation used for classification.

3.4.5. Node Anomaly Classification

Because this study focuses on binary log anomaly detection, the classification layer outputs the probability distribution over the normal and anomalous classes:

$$\hat{y}_i = \text{softmax}(W_c r_i + b_c) \quad (31)$$

where W_c and b_c are the parameters of the classification layer, and $\hat{y}_i \in \mathbb{R}^2$. Specifically, $\hat{y}_{i,0}$ denotes the probability that node v_i belongs to the normal class, and $\hat{y}_{i,1}$ denotes the probability that node v_i belongs to the anomalous class. The final predicted label is $\hat{y}_i = \arg \max_{c \in \{0,1\}} \hat{y}_{i,c}$.

3.5. Joint Loss Function

The training objective of CGATLog consists of three components: node anomaly classification, edge-relation prediction and anomalous-node generation. To achieve end-to-end collaborative optimization, we construct a joint loss function. For the labelled node set V_L , the node-classification loss is defined using cross-entropy:

$$L_{\text{cls}} = - \sum_{v_i \in V_L} \sum_{c=0}^1 \mathbf{1}(y_i = c) \log \hat{y}_{i,c} \quad (32)$$

where $\mathbf{1}(\cdot)$ is the indicator function, y_i is the ground truth label and $\hat{y}_{i,c}$ is the predicted probability that node v_i belongs to class c . The edge-prediction loss constrains the rationality of connections between generated nodes and the original graph structure, while enabling the edge generator to learn the topological distribution of the real graph:

$$\mathcal{L}_{\text{edge}} = - \frac{1}{|\mathcal{E}_c|} \sum_{(i,j) \in \mathcal{E}_c} [A_{ij} \log \hat{A}_{ij} + (1 - A_{ij}) \log (1 - \hat{A}_{ij})] \quad (33)$$

where $\hat{A}_{ij}$ is the edge probability output by the sigmoid predictor and A_{ij} is the corresponding binary adjacency label. This formulation aligns the optimization objective with the probabilistic output of the edge-relation predictor. The conditional adversarial generation loss drives the generator to learn the distribution of anomalous nodes in the embedding space:

$$L_{\text{cgan}} = \mathbb{E}_{z \sim p_{\text{data}}} [\log D_{\psi}(z, c_{\text{abn}})] + \mathbb{E}_{r \sim p_r} [\log (1 - D_{\psi}(G_{\theta}(r, c_{\text{abn}}), c_{\text{abn}}))] \quad (34)$$

Here, the generator G_{θ} attempts to minimize the generation term so that the generated anomalous node embeddings approach the distribution of real anomalous nodes, whereas the discriminator D_{ψ} attempts to distinguish real anomalous node embeddings from generated ones.

The optimization is performed in two stages. The CGAN is first pretrained using the minimax objective in Equation (18). After pretraining, the generated embeddings are incorporated into the graph, and the detection network and generator are optimized using the following weighted objective:

$$L_{\text{total}} = L_{\text{cls}} + \lambda_e L_{\text{edge}} + \lambda_g L_{\text{adv,G}}$$

where the two non-negative weighting coefficients control the contributions of edge-relation prediction and generator-side adversarial learning, respectively. The discriminator is updated alternately according to the adversarial objective, while the remaining modules minimize the total objective above.

3.6. Computational Complexity Analysis

Let N denote the number of log nodes in a processed graph partition, d the node embedding dimension, w the sliding window size, L the number of graph attention layers,

K the number of attention heads, T the maximum number of tokens per log entry, and h_B the BERT hidden dimension. Because each node is connected to at most $2w$ local neighbours, the number of candidate edges is $O(Nw)$. Semantic-similarity calculation and sparse edge prediction require $O(Nwd)$ operations, while the gated multi-head graph attention module requires $O(LKNwd)$. With the fixed window size $w = 5$ used in this study, these graph-related costs grow approximately linearly with N . BERT encoding has an attention cost of approximately $O(NT^2h_B)$ per Transformer layer, whereas the sequence-level Transformer retains quadratic self-attention with respect to its input sequence length. Therefore, CGATLog does not claim linear end-to-end complexity; its main scalability benefit comes from restricting graph construction and edge prediction to a bounded local neighbourhood.

4. Experiment Analysis

4.1. Datasets and Experimental Setup

The experiments are conducted on three publicly available benchmark datasets from LogHub [5]: HDFS, BGL, and Thunderbird. HDFS_v1 contains 11,175,629 log entries collected from a Hadoop Distributed File System over approximately 38.7 h. BGL contains 4,747,963 log entries collected from the Blue Gene/L supercomputer over approximately 214.7 days, while Thunderbird contains 211,212,192 log entries collected from the Thunderbird supercomputer over approximately 244 days. HDFS provides anomaly annotations at the block level, whereas BGL and Thunderbird provide alert-based labels associated with individual log entries. Since HDFS provides block-level labels, each log entry inherits the label of its corresponding block. These are derived node-level labels rather than independently annotated log entry labels.

Considering the substantially different scales of the three datasets and the computational cost of BERT-based graph construction, we use the complete HDFS_v1 dataset and chronologically select 1,000,000 log entries from BGL and 1,000,000 log entries from Thunderbird for the main experiments. For BGL and Thunderbird, the selected records form continuous chronological segments rather than randomly sampled individual entries, thereby preserving the original temporal dependencies. For HDFS, log entries are first grouped according to block identifiers, and the original block-level labels are retained during dataset partitioning.

To reduce the risk of temporal information leakage, all datasets are partitioned chronologically rather than randomly. The first 70% of each dataset is used for training, the subsequent 10% for validation, and the final 20% for testing. The graph structures of the training, validation, and test subsets are constructed independently. Therefore, the evaluation follows an inductive setting in which test nodes and test-set graph connections are not observed during model training. All hyperparameter selection and early stopping decisions are based exclusively on the validation set.

For BGL and Thunderbird, the original anomaly labels provided with the datasets are directly used as ground truth. For HDFS, the original block-level labels are used: all log entries belonging to the same block inherit the corresponding block label during preprocessing, while logs from the same block are kept within the same data partition to avoid cross-partition leakage. The default experiments retain the natural class distribution of each selected subset. In the class imbalance experiments in Section 4.5, only the training set is resampled to construct normal-to-anomalous ratios of 10:1, 50:1, 100:1, and 200:1, while the validation and test sets retain their original distributions.

4.2. Implementation Details

CGATLog is implemented in Python 3.10 using PyTorch 2.2 and CUDA 12.1. All experiments are conducted on a workstation equipped with an NVIDIA RTX 4090 GPU with 24 GB memory and 64 GB system memory. The Hugging Face Transformers library is used for BERT-based semantic encoding, and PyTorch Geometric is used to implement graph attention operations.

For semantic representation, BERT-base-uncased is adopted as the pretrained language encoder. It contains 12 Transformer layers with a hidden size of 768. The maximum token length of each log entry is set to 128. The output of BERT is projected to a $d = 256$ -dimensional node representation before being fed into the sequence-level Transformer. The additional Transformer encoder contains two layers and four attention heads with a feed-forward hidden dimension of 512 and a dropout rate of 0.1.

The graph attention anomaly detection module contains $L = 2$ graph attention layers, each with $K = 4$ attention heads. The hidden dimension is 256 and the dropout rate is 0.2. The semantic-temporal weighting coefficient used in graph construction is set to $\beta = 0.6$, which is selected from $\{0.2, 0.4, 0.6, 0.8\}$ using the validation set.

The generative augmentation module operates on 256-dimensional graph embeddings. The generator takes a 128-dimensional Gaussian noise vector concatenated with the anomaly condition embedding as input and uses three fully connected layers with dimensions 256, 512, and 256. ReLU activation and batch normalization are used in the hidden layers. The discriminator consists of three fully connected layers with dimensions 256, 256, and 64, followed by a sigmoid output layer. LeakyReLU with a negative slope of 0.2 is used in the discriminator, and dropout is set to 0.2. Binary cross-entropy is used as the adversarial loss.

The CGAN is first trained separately for 100 epochs with a batch size of 128. The generator and discriminator are updated at a 1:1 ratio using Adam with a learning rate of 2×10^{-4} and $(\beta_1, \beta_2) = (0.5, 0.999)$. To improve practical training stability, the generator uses batch normalization, the discriminator uses dropout, and the two networks are updated alternately at a 1:1 ratio after separate CGAN pretraining. These settings were used to reduce rapid discriminator domination and unstable gradient updates. However, no explicit gradient-penalty or spectral-normalization regularizer is employed in the current implementation.

After CGAN pretraining, the generated anomalous embeddings are incorporated into the graph and the complete detection model is optimized jointly. The oversampling number M is dynamically determined so that the number of generated anomalous nodes increases the anomalous class in the training set to 50% of the number of normal nodes, i.e., $M = \max(0, \lfloor 0.5N_{\text{normal}} \rfloor - N_{\text{anomaly}})$.

No generated samples are added to the validation or test sets.

For the complete CGATLog model, AdamW is used with an initial learning rate of 2×10^{-5} for BERT and 1×10^{-3} for the remaining modules. The batch size is 32 and the model is trained for at most 50 epochs. A linear learning-rate schedule with a warmup ratio of 0.1 is used for BERT. Early stopping is applied when the validation F1-score does not improve for eight consecutive epochs. Gradient clipping with a maximum norm of 1.0 is employed to improve training stability.

Hyperparameters are determined using grid search on the validation set. The main search spaces are $d \in \{128, 256, 512\}$, $L \in \{1, 2, 3\}$, $K \in \{2, 4, 8\}$, $\beta \in \{0.2, 0.4, 0.6, 0.8\}$, graph attention dropout in $\{0.1, 0.2, 0.3\}$, and CGAN augmentation targets in $\{0.25, 0.50, 0.75, 1.00\}$ times the number of normal training samples. The configuration giving the highest validation F1-score is used for the final experiments.

4.3. Performance Testing Indicators

We compare CGATLog with representative baselines, including BiLSTM, Attention-LSTM, CNN-BiLSTM, Transformer Encoder, LogBERT [38], DeepLog [39], LAnoBERT [30], MidLog [31], LogGT, and LogLLM [32]. Table 1 reports Precision, Recall, and F1-score on HDFS, Thunderbird, and BGL. CGATLog achieves F1-scores of 98.00%, 98.35%, and 98.22%, respectively. The model maintains consistently high Precision and Recall across the three datasets. Although MidLog achieves a slightly higher F1-score on HDFS (98.10%), CGATLog provides more stable performance across all datasets.

Table 1. Overall detection performance on the BGL, Thunderbird and HDFS datasets (%).

Method	BGL			Thunderbird			HDFS		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
BiLSTM	94.62	93.85	94.23	92.87	94.16	93.51	95.36	91.74	93.51
Attention-LSTM	95.24	94.71	94.97	93.56	95.03	94.29	96.08	92.63	94.32
CNN-BiLSTM	96.40	95.23	95.81	94.91	96.28	95.59	98.32	83.67	90.41
Transformer Encoder	96.31	95.76	96.03	95.68	96.14	95.91	96.57	93.91	95.22
LogBERT	89.54	92.39	90.94	96.75	96.52	96.63	87.01	78.13	82.33
DeepLog	89.74	82.78	86.12	87.34	99.61	93.07	88.44	69.49	77.83
LAnoBERT	90.80	92.60	91.70	91.70	95.80	93.70	93.50	98.20	95.80
MidLog	97.40	98.70	98.00	94.90	98.20	96.50	97.20	99.10	98.10
LogLLM	92.90	97.20	95.00	99.80	66.50	80.00	98.70	93.80	96.30
LogGT	98.30	98.10	98.60	98.10	98.20	98.15	99.50	99.30	99.60
CGATLog	98.37	98.08	98.22	98.24	98.47	98.35	98.18	97.83	98.00

As illustrated in Figure 3, the results also highlight the limitations of conventional sequence models. BiLSTM, Attention-LSTM, and DeepLog generally achieve lower F1-scores than CGATLog. On HDFS, DeepLog attains only 69.49% Recall, while CNN-BiLSTM reaches 98.32% Precision but only 83.67% Recall. Transformer Encoder is more stable but does not explicitly model graph structure, temporal intervals, or event associations. Other baselines also vary considerably across datasets; for example, LogLLM achieves 99.80% Precision but only 66.50% Recall on Thunderbird. In contrast, CGATLog combines semantic representations, temporal structure, graph relations, and anomaly sample augmentation, resulting in more balanced performance.

4.4. Ablation Study

We evaluate the contribution of each CGATLog component using eight ablation variants. CGATLog-BERT replaces BERT representations with randomly initialized word embeddings. CGATLog-Transformer removes the sequence-level Transformer. CGATLog-CGAN disables anomaly sample augmentation. CGATLog-EdgePred replaces learned edge prediction with random connections for generated nodes. CGATLog-Time removes temporal features. CGATLog-Gate replaces adaptive semantic-temporal gating with fixed weights. CGATLog-ResLN removes residual connections and layer normalization. CGATLog-GAT replaces the proposed graph attention module with a standard GAT. Table 2 reports the results, and Figure 4 visualizes F1-score changes.

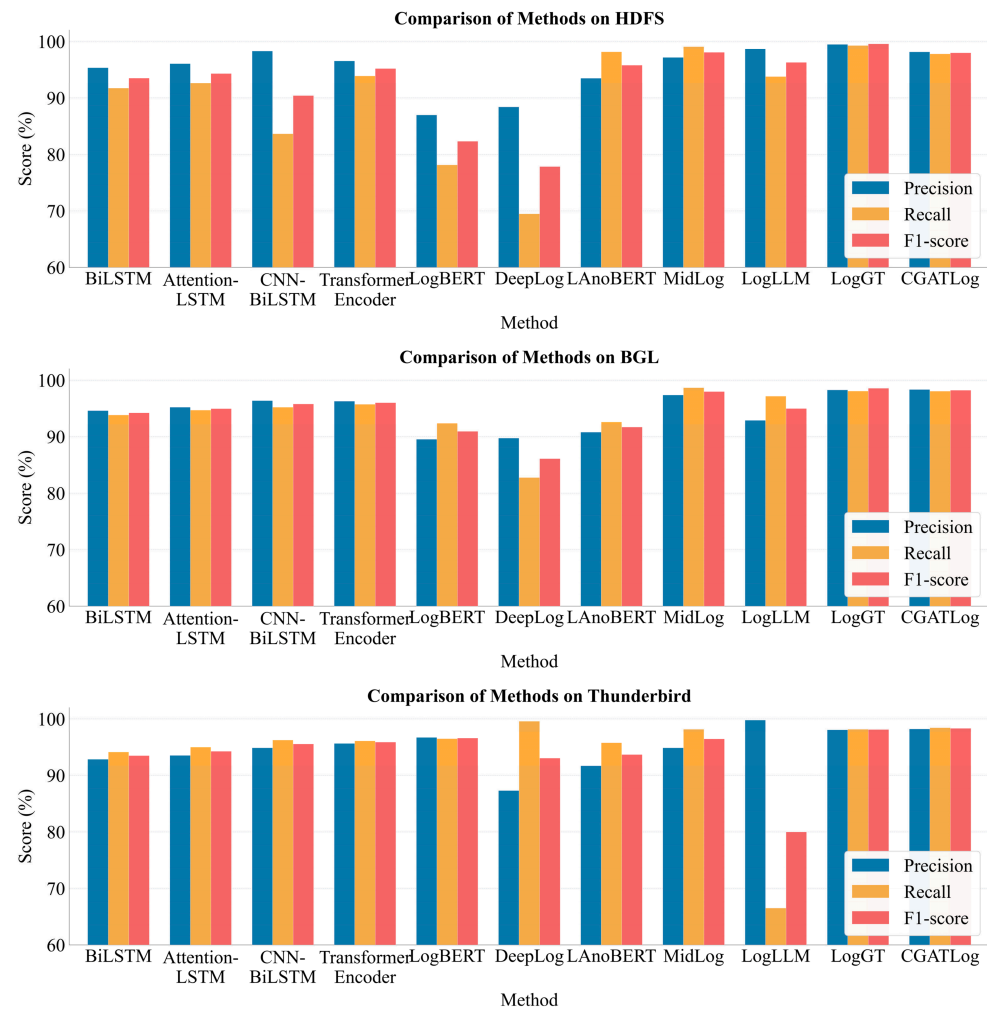


Figure 3. Detection performance comparison on the BGL, Thunderbird and HDF5 datasets.

Table 2. Ablation study results on the BGL, Thunderbird and HDF5 datasets (%).

Method	BGL			Thunderbird			HDFS		
	Precision	Recall	F1-Score	Precision	Recall	F1-Score	Precision	Recall	F1-Score
CGATLog	98.37	98.08	98.22	98.24	98.47	98.35	98.18	97.83	98.00
CGATLog-BERT	89.46	86.73	87.95	88.92	87.38	88.06	90.15	85.64	87.43
CGATLog-Transformer	93.82	91.76	92.63	92.67	91.48	91.96	94.08	90.32	91.84
CGATLog-CGAN	94.25	87.91	90.36	93.86	88.45	90.74	94.73	86.92	89.51
CGATLog-EdgePred	93.61	89.34	91.18	93.24	89.86	91.31	94.12	88.17	90.62
CGATLog-Time	95.87	93.42	94.48	95.31	93.86	94.42	95.74	92.67	93.96
CGATLog-Gate	96.42	94.75	95.47	96.13	94.92	95.39	96.38	93.84	94.87
CGATLog-ResLN	97.26	96.18	96.64	97.08	96.47	96.69	97.11	95.76	96.31
CGATLog-GAT	95.64	93.08	94.19	95.21	93.57	94.23	95.37	92.41	93.61

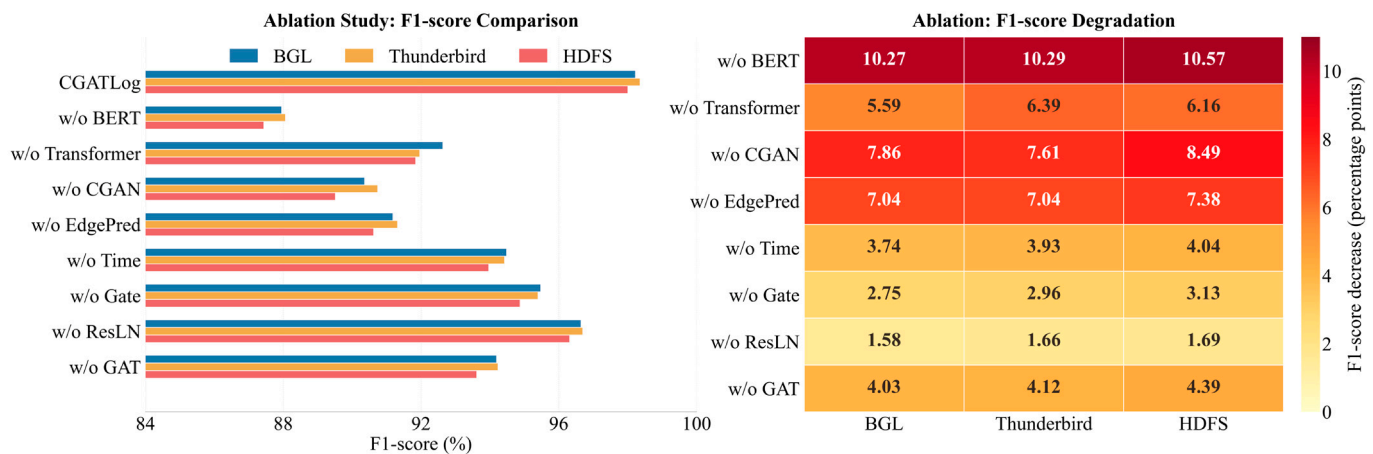


Figure 4. Ablation study results of CGATLog variants.

As shown in Table 2 and Figure 4, the BERT semantic encoder had the most substantial effect on model performance. After removing BERT, the average F1-score across the three datasets decreased by 10.38 percentage points, representing the most severe degradation among all ablation variants. This result indicates that pretrained language models provide indispensable contextual semantic representations for log nodes. Notably, removing BERT led to a larger decrease in Recall, averaging 11.54 percentage points, than in Precision, which decreased by 8.75 percentage points. This suggests that BERT is particularly important for detecting anomalous logs, because anomalies often involve rare semantic combinations that require deep contextual modelling for effective recognition.

For class balancing enhancement, the contributions of the CGAN and edge-relation prediction modules were mainly reflected in Recall. After removing the CGAN, Recall decreased by an average of 10.37 percentage points, whereas Precision decreased by only 3.98 percentage points. This asymmetric effect is consistent with the design objective of the CGAN, which supplements minority anomaly samples and prevents the decision boundary from being biased towards the majority normal class. After removing edge-relation prediction, the average F1-score decreased by 7.15 percentage points, indicating that reasonably integrating generated nodes into the graph topology is essential for preserving graph structure integrity. Random connections introduce noise and weaken the benefit of anomaly sample augmentation.

For graph structure modelling, temporal features and the gated fusion mechanism showed complementary effects. Removing temporal features reduced the average F1-score by 3.90 percentage points, with the largest effect observed on HDFS, where the decrease reached 4.04 percentage points. This is likely because normal read and write operations in HDFS follow relatively stable temporal rhythms, whereas anomalous events are often accompanied by shifts in time intervals. Removing the gated fusion mechanism led to an average decrease of 2.95 percentage points, showing that adaptively adjusting the relative contributions of semantic and temporal information is more effective than using fixed-weight fusion.

In addition, removing the sequence-level Transformer reduced the average F1-score by 6.05 percentage points, confirming the importance of contextual dependency modelling at the log-sequence level. Replacing the improved graph attention module with a standard GAT resulted in an average F1-score decrease of 4.18 percentage points. Under the sliding window graph construction with $w = 5$, each node can interact with at most $2w = 10$ neighbouring nodes, providing richer local connectivity than a simple sequence chain. A standard GAT calculates attention mainly from node features when aggregating neighbouring information. In contrast, the proposed graph attention module

adaptively fuses semantic and temporal association scores through the gating mechanism in Equations (23)–(27), assigning differentiated weights to different node pairs according to both content relevance and temporal patterns. Therefore, the observed performance improvement is mainly attributed to the semantic–temporal gated attention design rather than to graph topology alone. This mechanism is complementary to the sequence-level Transformer: the Transformer captures long-range dependencies along the log sequence, whereas the gated graph attention models fine-grained pairwise semantic–temporal interactions within the local window. The effect of removing residual connections and layer normalization was the smallest, with an average F1-score decrease of 1.64 percentage points, suggesting that these components mainly serve to stabilize training rather than directly drive the primary performance gain.

4.5. Performance Under Different Class Imbalance Ratios

We further evaluate CGATLog under normal-to-anomalous training ratios of 10:1, 50:1, 100:1, and 200:1. The validation and test sets retain their original distributions. For each ratio, all anomalous training samples are retained and normal samples are randomly subsampled. The same resampled training subset is used for every compared method. We report Recall and F1-score for BiLSTM, Attention-LSTM, CNN-BiLSTM, Transformer Encoder, LogBERT, DeepLog, LAnoBERT [30], MidLog [31], LogLLM [32], and CGATLog on BGL, Thunderbird, and HDFS.

As shown in Tables 3–5, Recall and F1-score decreased for all methods as the imbalance ratio increased, indicating that anomaly sample scarcity substantially impairs the ability of models to recognize minority anomalous behaviours. Among the baselines, DeepLog, LogBERT and LogLLM showed particularly pronounced performance degradation under high imbalance ratios, suggesting that models relying mainly on sequential patterns or language-model representations are vulnerable to shifts in sample distribution. By contrast, CGATLog consistently achieved the highest F1-score on BGL and Thunderbird, and maintained near-optimal detection performance on HDFS, demonstrating strong robustness across different log scenarios. Figures 5 and 6 further illustrate the F1-score and Recall trends, respectively, under different class imbalance ratios.

Table 3. Performance under different imbalance ratios on the BGL dataset (%).

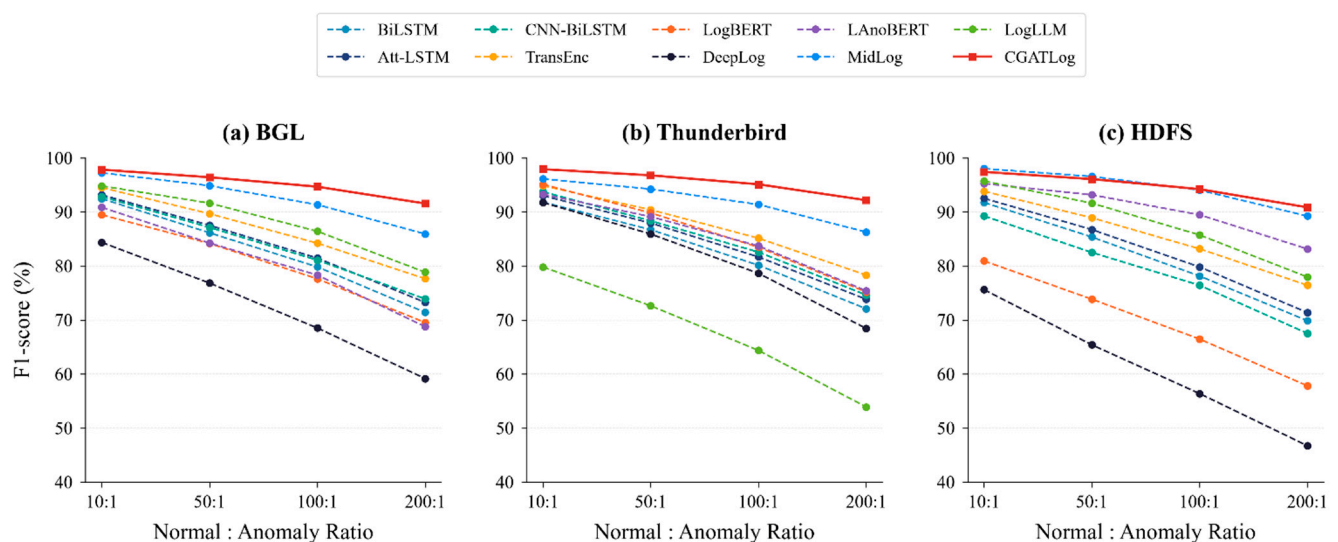
	10:1		50:1		100:1		200:1	
Method	Recall	F1	Recall	F1	Recall	F1	Recall	F1
BiLSTM	91.38	92.46	83.27	86.14	74.52	79.83	63.18	71.42
Attention-LSTM	92.14	93.18	84.63	87.52	76.38	81.47	65.72	73.26
CNN-BiLSTM	93.07	92.85	85.91	87.13	77.64	81.08	67.35	73.91
Transformer Encoder	93.86	94.52	87.42	89.67	80.15	84.23	71.83	77.64
LogBERT	90.26	89.47	82.75	84.18	73.41	77.62	61.87	69.53
DeepLog	80.93	84.37	72.16	76.83	62.74	68.52	51.38	59.16
LAnoBERT	89.74	90.86	81.63	84.25	72.85	78.31	60.94	68.76
MidLog	96.83	97.25	93.74	94.86	89.62	91.35	82.47	85.93
LogLLM	95.61	94.82	90.28	91.64	83.75	86.42	73.96	78.85
CGATLog	97.52	97.83	95.87	96.42	93.24	94.68	89.63	91.57

Table 4. Performance under different imbalance ratios on the Thunderbird dataset (%).

	10:1		50:1		100:1		200:1	
Method	Recall	F1	Recall	F1	Recall	F1	Recall	F1
BiLSTM	92.07	91.83	84.51	86.72	75.86	80.14	64.37	72.08
Attention-LSTM	93.28	93.02	85.74	87.95	77.23	81.68	66.52	73.84
CNN-BiLSTM	94.36	93.75	87.18	88.36	79.42	82.57	68.91	74.63
Transformer Encoder	94.52	94.87	88.36	90.41	81.27	85.14	72.64	78.32
LogBERT	94.73	95.12	87.92	89.83	79.86	83.47	68.24	75.18
DeepLog	98.14	91.76	92.37	85.92	83.51	78.64	71.28	68.43
LAnoBERT	94.85	93.28	87.64	89.13	80.25	83.72	69.36	75.41
MidLog	97.34	96.12	94.86	94.25	90.71	91.38	83.64	86.29
LogLLM	66.28	79.82	58.73	72.64	49.85	64.37	38.62	53.91
CGATLog	97.86	97.92	96.13	96.78	93.87	95.12	90.42	92.18

Table 5. Performance under different imbalance ratios on the HDFS dataset (%).

	10:1		50:1		100:1		200:1	
Method	Recall	F1	Recall	F1	Recall	F1	Recall	F1
BiLSTM	89.83	91.76	81.42	85.37	72.68	78.14	61.53	69.87
Attention-LSTM	90.74	92.53	82.86	86.74	74.15	79.82	63.47	71.38
CNN-BiLSTM	81.92	89.26	74.38	82.51	66.27	76.43	56.84	67.52
Transformer Encoder	92.17	93.84	85.73	88.92	78.46	83.18	69.82	76.43
LogBERT	76.28	80.93	68.42	73.86	59.73	66.47	49.16	57.82
DeepLog	67.83	75.62	58.27	65.41	48.91	56.38	38.64	46.72
LAnoBERT	97.62	95.26	93.74	93.18	88.63	89.47	80.52	83.16
MidLog	98.73	98.02	96.42	96.57	92.91	94.05	86.35	89.24
LogLLM	93.16	95.72	87.45	91.63	80.38	85.74	70.51	77.96
CGATLog	97.14	97.42	95.36	96.08	92.78	94.23	88.92	90.86

**Figure 5.** F1-score trends under different normal-to-anomalous imbalance ratios.

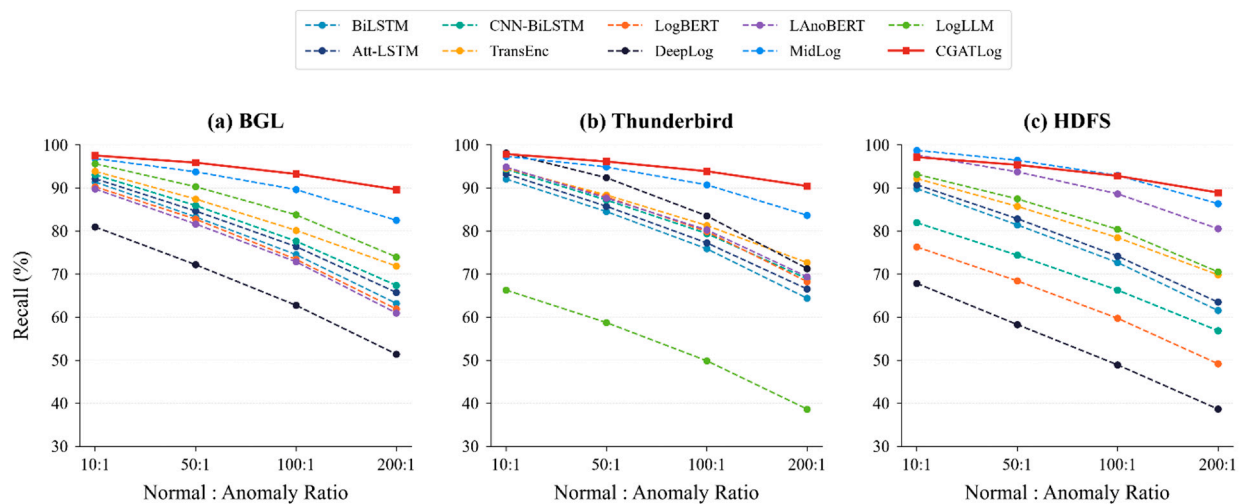


Figure 6. Recall trends under different normal-to-anomalous imbalance ratios.

A closer analysis of the severely imbalanced setting further confirms this advantage. When the ratio of normal to anomalous logs reached 200:1, CGATLog achieved F1-scores of 91.57%, 92.18% and 90.86% on BGL, Thunderbird and HDFS, respectively, outperforming all compared methods. Although MidLog slightly outperformed CGATLog on HDFS under the 10:1 and 50:1 settings, which is consistent with its strong performance on HDFS in the overall comparison, CGATLog gradually gained the advantage as the imbalance became more severe, particularly under the 100:1 and 200:1 settings. Figure 7 further illustrates the Recall and F1-score differences between CGATLog and the best-performing baseline across different imbalance ratios. This result suggests that the conditional generative adversarial network in CGATLog can enhance minority anomalous node representations in the feature space and mitigate class bias caused by insufficient anomaly samples. In addition, graph structure modelling and semantic-temporal association fusion help the model capture latent relationships among sparse anomalous behaviours, thereby maintaining more stable anomaly recognition capability under extreme class imbalance.

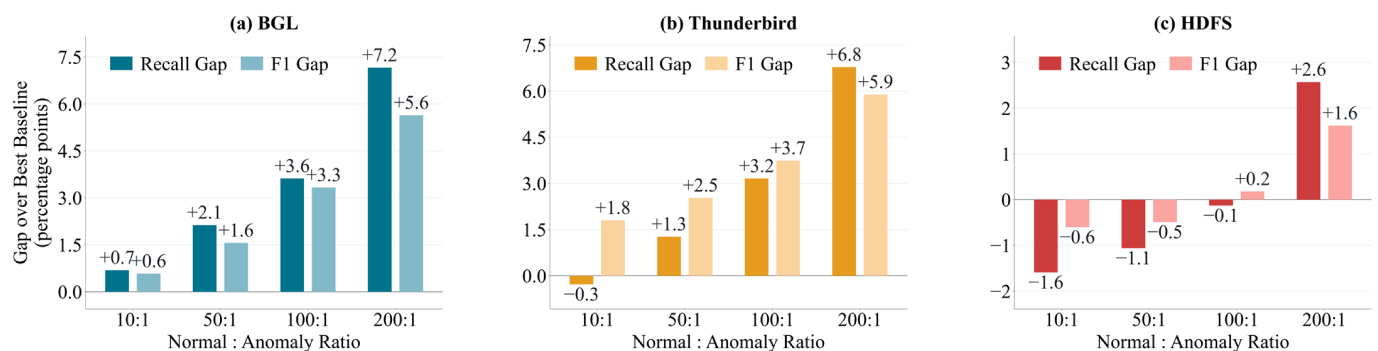


Figure 7. Performance advantage of CGATLog over the best baseline under different imbalance ratios.

4.6. Computational Efficiency and Training Stability

To complement the accuracy-oriented evaluation, we further compare the empirical computational cost of representative methods under the same hardware environment described in Section 4.2. Training time, inference time, and peak GPU memory consumption are reported in Table 6. The results indicate that CGATLog incurs additional training overhead because of BERT-based semantic encoding, graph construction, and generative augmentation, while maintaining a practical inference cost.

Table 6. Computational efficiency comparison on the BGL dataset.

Peak GPU Memory (GB)	Inference Time (s)	Training Time (min)	Method
3.2	21.4	38.6	BiLSTM
5.6	28.7	54.8	Transformer
10.4	41.2	88.3	Encoder
6.8	33.8	61.5	LogBERT
13.6	46.9	112.7	MidLog
			CGATLog

Because CGAN optimization is stochastic, we further evaluate the training stability of CGATLog using five independent random seeds (42, 52, 62, 72, and 82). Table 7 summarizes the mean F1-score, standard deviation, and variance. The small dispersion across runs suggests that the stochastic generation process does not introduce substantial performance fluctuation.

Table 7. F1-score stability across multiple random seeds.

Variance	Std. (%)	Mean F1-Score (%)	Dataset
0.0121	0.11	98.22	BGL
0.0081	0.09	98.35	Thunderbird
0.0169	0.13	98.00	HDFS

5. Conclusions

This study proposes CGATLog, a graph-learning framework for anomaly detection in heterogeneous, temporally ordered system logs. The framework represents log observations as a weighted graph that integrates semantic content, event-execution relationships, and temporal intervals. A BERT-Transformer encoder extracts contextual semantic features, while a conditional generative adversarial network augments minority anomalous node representations in the graph embedding space. A semantic-temporal gated graph attention mechanism then performs node-level log anomaly classification. Experiments on HDFS, BGL, and Thunderbird yield F1-scores of 98.00%, 98.22%, and 98.35%, respectively, demonstrating competitive and stable performance. Ablation results show that BERT-based semantic encoding and CGAN-based anomaly augmentation make the largest contributions. Under severe class imbalance, CGATLog maintains Recall above 88%, while most baselines fall below 70%. These results support the effectiveness of graph embedding augmentation and adaptive semantic-temporal relation modelling. The current evaluation is limited to public system-log benchmarks; application to real-time IoT or streaming environments still requires domain-specific validation. In addition, the BERT-Transformer architecture introduces substantial computational overhead, and the standard adversarial objective may remain sensitive to training instability and mode collapse.

Author Contributions: Conceptualization, S.G. and X.W.; methodology, S.G.; writing—original draft preparation, S.G.; writing—review and editing, S.G. and X.W.; visualization, S.G.; supervision, S.G.; project administration, S.G. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Data Availability Statement: The HDFS, BGL, and Thunderbird datasets analyzed in this study are publicly available from the LogHub repository at <https://github.com/logpai/loghub> (accessed on 30 September 2026). The dataset collection is documented in [5].

Acknowledgments: The authors have reviewed and edited the output and take full responsibility for the content of this publication.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Nassereddine, M.; Khang, A. Applications of Internet of Things (IoT) in smart cities. In *Advanced IoT Technologies and Applications in the Industry 4.0 Digital Economy*; CRC Press: Boca Raton, FL, USA, 2024; pp. 109–136.
2. Xu, W.; Huang, L.; Fox, A.; Patterson, D.; Jordan, M.I. Detecting large-scale system problems by mining console logs. In *Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP)*; Association for Computing Machinery: New York, NY, USA, 2009; pp. 117–132.
3. Oliner, A.; Stearley, J. What supercomputers say: A study of five system logs. In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*, Edinburgh, UK, 25–28 June 2007; pp. 575–584.
4. Alsalamah, H.A.; Ismail, W.N. Evolutionary computation for feature optimization and image-based dimensionality reduction in IoT intrusion detection. *Mathematics* **2025**, *13*, 3869. [\[CrossRef\]](#)
5. Zhu, J.; He, S.; He, P.; Liu, J.; Lyu, M.R. Loghub: A large collection of system log datasets for AI-driven log analytics. In *Proceedings of the IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, Florence, Italy, 9–12 October 2023; pp. 355–366.
6. Alzamil, A.; Guesmi, R.; Alghadhbani, A.; Guesmi, T.; Kachouri, A. DualBERT: Fusing symbolic and temporal dynamics for high-precision log anomaly detection. *IEEE Access* **2026**, *14*, 11369–11384. [\[CrossRef\]](#)
7. He, P.; Zhu, J.; Zheng, Z.; Lyu, M.R. Drain: An online log parsing approach with fixed depth tree. In *Proceedings of the 2017 IEEE International Conference on Web Services (ICWS)*, Honolulu, HI, USA, 25–30 June 2017; pp. 33–40.
8. Bai, Z.; Miao, H.; Miao, J.; Xiao, N.; Sun, X. Artificial intelligence-driven cybersecurity applications and challenges. *Innov. Appl. AI* **2025**, *2*, 26–33. [\[CrossRef\]](#)
9. Hadadi, F.; Xu, Q.; Bianculli, D.; Briand, L. LLM meets ML: Data-efficient anomaly detection on unstable logs. *ACM Trans. Softw. Eng. Methodol.* **2025**, *35*, 1–48. [\[CrossRef\]](#)
10. Zhang, X.; Xu, Y.; Lin, Q.; Qiao, B.; Zhang, H.; Dang, Y.; Xie, C.; Yang, X.; Cheng, Q.; Li, Z.; et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*; Association for Computing Machinery: New York, NY, USA, 2019; pp. 807–817.
11. Gao, Y.; Luo, T.; Huang, K.; Tang, J.; Li, X. LogLAA: An adaptive integrated log anomaly analysis framework. *Cybersecurity* **2026**, *9*, 141. [\[CrossRef\]](#)
12. Le, V.-H.; Zhang, H. Log-based anomaly detection without log parsing. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, Melbourne, Australia, 15–19 November 2021; pp. 492–504.
13. Guo, H.; Yang, J.; Liu, J.; Bai, J.; Wang, B.; Li, Z.; Zheng, T.; Zhang, B.; Peng, J.; Tian, Q. LogFormer: A pre-train and tuning pipeline for log anomaly detection. *Proc. AAAI Conf. Artif. Intell.* **2024**, *38*, 135–143. [\[CrossRef\]](#)
14. Almodovar, C.; Sabrina, F.; Karimi, S.; Azad, S. LogFiT: Log anomaly detection using fine-tuned language models. *IEEE Trans. Netw. Serv. Manag.* **2024**, *21*, 1715–1723. [\[CrossRef\]](#)
15. Huang, H.; Luo, W.; Wang, Y.; Zhou, Y.; Huang, W. LogCTBL: A hybrid deep learning model for log-based anomaly detection. *J. Supercomput.* **2025**, *81*, 448. [\[CrossRef\]](#)
16. Li, Z.; Shi, J.; van Leeuwen, M. Graph neural networks based log anomaly detection and explanation. *Data Min. Knowl. Discov.* **2026**, *40*, 66. [\[CrossRef\]](#) [\[PubMed\]](#)
17. Wang, P.; Zhang, X.; Cao, Z.; Xu, W.; Li, W. LogGT: Cross-system log anomaly detection via heterogeneous graph feature and transfer learning. *Expert Syst. Appl.* **2024**, *251*, 124082. [\[CrossRef\]](#)
18. Yilmaz, A.; Das, R. A novel hybrid approach combining GCN and GAT for effective anomaly detection from firewall logs in campus networks. *Comput. Netw.* **2025**, *259*, 111082. [\[CrossRef\]](#)
19. Zhang, J.; Luo, C.; Shao, R. MTG-GAN: A masked temporal graph generative adversarial network for cross-domain system log anomaly detection. *IEEE Trans. Netw. Serv. Manag.* **2026**, early access. [\[CrossRef\]](#)
20. Pan, J.; Wong, S.L.; Yuan, Y. RAGLog: Log anomaly detection using retrieval augmented generation. In *Proceedings of the 2024 IEEE World Forum on Public Safety Technology (WFPST)*, Herndon, VA, USA, 14–15 May 2024; pp. 169–174.
21. Duan, C.; Jia, T.; Yang, Y.; Liu, G.; Liu, J.; Zhang, H.; Zhou, Q.; Li, Y.; Huang, G. EagerLog: Active learning enhanced retrieval augmented generation for log-based anomaly detection. In *Proceedings of the 2025 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Hyderabad, India, 6–11 April 2025.

22. Zhang, L.; Jia, T.; Jia, M.; Wu, Y.; Liu, H.; Li, Y. XRAGLog: A resource-efficient and context-aware log-based anomaly detection method using retrieval-augmented generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*; Association for the Advancement of Artificial Intelligence: Washington, DC, USA, 2025.
23. He, M.; Jia, T.; Duan, C.; Cai, H.; Li, Y.; Huang, G. LLMeLog: An approach for anomaly detection based on LLM-enriched log events. In *Proceedings of the IEEE 35th International Symposium on Software Reliability Engineering (ISSRE)*, Tsukuba, Japan, 28–31 October 2024; pp. 132–143.
24. Yang, Z.; Harris, I.G. LogLLaMA: Transformer-based log anomaly detection with LLaMA. In *Proceedings of the 2025 International Joint Conference on Neural Networks (IJCNN)*, Rome, Italy, 30 June–5 July 2025.
25. He, M.; Jia, T.; Duan, C.; Cai, H.; Li, Y.; Huang, G. Weakly-supervised log-based anomaly detection with inexact labels via multi-instance learning. In *Proceedings of the IEEE/ACM 47th International Conference on Software Engineering (ICSE)*, Ottawa, ON, Canada, 26 April–6 May 2025; pp. 2918–2930.
26. Wurzenberger, M.; Höld, G.; Landauer, M.; Skopik, F. Analysis of statistical properties of variables in log data for advanced anomaly detection in cyber security. *Comput. Secur.* **2024**, *137*, 103631. [\[CrossRef\]](#)
27. Zhang, C.; Jia, T.; Shen, G.; Zhu, P.; Li, Y. MetaLog: Generalizable cross-system anomaly detection from logs with meta-learning. In *Proceedings of the 46th IEEE/ACM International Conference on Software Engineering (ICSE)*, Lisbon Portugal, 14–20 April 2024; p. 154.
28. Fei, K.; Zhou, J.; Zhou, Y.; Gu, X.; Fan, H.; Li, B.; Wang, W.; Chen, Y. LaAeb: A comprehensive log-text analysis based approach for insider threat detection. *Comput. Secur.* **2025**, *148*, 104126. [\[CrossRef\]](#)
29. Farjamfard, E.; Shamel-Sendi, A. Enhanced anomaly detection in log-based systems using GAN-based models. *Clust. Comput.* **2025**, *28*, 720. [\[CrossRef\]](#)
30. Lee, Y.; Kim, J.; Kang, P. LAnoBERT: System log anomaly detection based on BERT masked language model. *Appl. Soft Comput.* **2023**, *146*, 110689. [\[CrossRef\]](#)
31. Yuan, W.; Ying, S.; Duan, X.; Cheng, H.; Zhao, Y.; Shang, J. MidLog: An automated log anomaly detection method based on multi-head GRU. *J. Syst. Softw.* **2025**, *226*, 112431. [\[CrossRef\]](#)
32. Guan, W.; Cao, J.; Qian, S.; Gao, J.; Ouyang, C. LogLLM: Log-based anomaly detection using large language models. *arXiv* **2024**, arXiv:2411.08561.
33. Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A.N.; Kaiser, Ł.; Polosukhin, I. Attention is all you need. In *Proceedings of the 31st Conference on Neural Information Processing Systems (NIPS 2017)*, Long Beach, CA, USA, 4–9 December 2017; pp. 6000–6010.
34. Devlin, J.; Chang, M.-W.; Lee, K.; Toutanova, K. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (NAACL-HLT)*; Association for Computational Linguistics: Stroudsburg, PA, USA, 2019; pp. 4171–4186.
35. Veličković, P.; Cucurull, G.; Casanova, A.; Romero, A.; Lio, P.; Bengio, Y. Graph attention networks. *arXiv* **2018**, arXiv:1710.10903.
36. Goodfellow, I.J.; Pouget-Abadie, J.; Mirza, M.; Xu, B.; Warde-Farley, D.; Ozair, S.; Courville, A.; Bengio, Y. Generative adversarial nets. In *Proceedings of the 27th International Conference on Neural Information Processing Systems*, Montreal, QC, Canada, 8–13 December 2014; pp. 2672–2680.
37. Mirza, M.; Osindero, S. Conditional generative adversarial nets. *arXiv* **2014**, arXiv:1411.1784.
38. Guo, H.; Yuan, S.; Wu, X. LogBERT: Log anomaly detection via BERT. In *Proceedings of the 2021 International Joint Conference on Neural Networks (IJCNN)*, Shenzhen, China, 18–22 July 2021; pp. 1–8.
39. 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*; Association for Computing Machinery: New York, NY, USA, 2017; pp. 1285–1298.

Disclaimer/Publisher’s Note: The statements, opinions and data contained in all publications are solely those of the individual author(s) and contributor(s) and not of MDPI and/or the editor(s). MDPI and/or the editor(s) disclaim responsibility for any injury to people or property resulting from any ideas, methods, instructions or products referred to in the content.