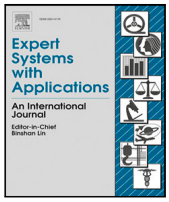
Expert Systems With Applications 251 (2024) 124082

Contents lists available at ScienceDirect

Expert Systems With Applications

journal homepage: www.elsevier.com/locate/eswa



LogGT: Cross-system log anomaly detection via heterogeneous graph feature and transfer learning

Peipeng Wang, Xiuguo Zhang *, Zhiying Cao, Weigang Xu, Wangwang Li

Information Science and Technology College, Dalian Maritime University, Dalian, Liaoning, China

ARTICLE INFO

Keywords:
Heterogeneous graph
Cross-system
Normalizing flow
Time intervals
Transfer learning

ABSTRACT

Automated system log anomaly detection plays a crucial role in ensuring service reliability. Existing methods incompletely utilize structured log entries, resulting in the loss of key information such as components and time. Besides, due to the limitations of labeled data, models trained by a single system are difficult to apply to other systems. Therefore, we propose a cross-system log anomaly detection method named LogGT, which simultaneously models log events, components and time, leveraging labeled system knowledge to achieve anomaly detection in unlabeled systems. Specifically, we firstly design a heterogeneous graph to accurately represent the interactions between different events and components in the log sequence. Then, in order to avoid noise interference and conduct cross-system semantic analysis, we employ BERT to extract log sentence vectors, and Normalizing Flow is used to optimize them for smoother node embedding. A Graph Transformer Architecture with Time Intervals (GTAT) is proposed to model heterogeneous graphs by integrating time feature, allowing for a comprehensive analyze of execution order and time anomalies. Additionally, we design a semantic weighting method and utilize a novel domain-adapted transfer learning technology to effectively transfer the heterogeneous graph features of the source system to the target system. Experimental results demonstrate that LogGT outperforms five log anomaly detection methods, achieving an average anomaly detection F1-score higher than 0.95. Moreover, the AUC value of GTAT exceeds the sequence model by more than 2.3%.

1. Introduction

The status, behavior, and other key information of large software systems are typically recorded in logs. Analyzing these logs can effectively ensure the stable operation of the system (He et al., 2021). The log, as a typical semi-structured text, includes not only the process status but also time, level, components, and more. Only through a comprehensive analysis of the complex log messages can we accurately predict system failures.

Existing mainstream log anomaly detection methods mainly utilize deep learning models to detect anomalies through supervised or unsupervised learning. In the unsupervised learning method, DeepLog (Du, Li, Zheng, & Srikumar, 2017) used the template index as input, learned the normal log sequence pattern through LSTM. However, this index-based approach ignores the semantics of words and sentences within the template, leading to a lack of robustness in the model. To address this issue, LogAnomaly (Meng et al., 2019) utilized word embedding to obtain template semantics. Nevertheless, unsupervised learning methods have been suffering from low accuracy due to a lack of knowledge about abnormalities. Among the supervised learning methods, LogRobust (Zhang et al., 2019) employed an attention-based Bi-LSTM to detect anomalies. NeuralLog (Le & Zhang, 2022b) utilized Transformer to capture semantic information in log sequences. However, from the perspective of logs, system anomalies are primarily manifested through changes in the execution order of logs and the time intervals between logs. The aforementioned method disregards the component and time information in the log during anomaly detection, which can inevitably overlook certain anomalies or lead to incorrect assessments.

First, the software system component records some additional contextual information such as the source and location of log events. Some system failures can cause workflow changes between different components. As shown in Table 1, in the Thunderbird dataset, components such as “Xinetd”, “Syslog” and “Dataeng”, etc. represent different protocols and processes, and contain different log events. A subsequence corresponding to these events, such as Eseq1 “E13→E13”, may correspond to different subsequences of components, Cseq1, Cseq2, or Cseq3. If the model does not clearly identify the corresponding

* Corresponding author.
E-mail addresses: wpp7@dlmu.edu.cn (P. Wang), zhangxg@dlmu.edu.cn (X. Zhang), czysophy@dlmu.edu.cn (Z. Cao), xu_wg@dlmu.edu.cn (W. Xu), 1120211468@dlmu.edu.cn (W. Li).

https://doi.org/10.1016/j.eswa.2024.124082
Received 20 December 2023; Received in revised form 15 April 2024; Accepted 19 April 2024
Available online 20 April 2024

Table 1
Some components, log events and subsequence in Thunderbird.

Components (ID)	Templates (ID)	Sub-sequence
Xinetd (C11)	f8c00723 (E76), 2315719c (E13), 73032e3b (E318), 9eb8e3c7 (E319), 49f6bba5 (E7), 6208f537 (E320)	Eseq1: ...E13, E13...;
Syslog (C15)	2315719c (E13), abd14c95 (E256), 47adbf55 (E87)	Cseq1...C46, C43...;
Dataeng (C43)	2315719c (E13)	Cseq2...C11, C15...;
haldaemon (C46)	2315719c (E13)	Cseq3...C38, C38...

components, the actual Cseq1 may be recognized as Cseq2 or Cseq3, resulting in misjudgment. For this reason, LogC (Yin et al., 2020) models both components and event sequences, and uses a combined LSTM to analyze anomalies in component sequences. However, this independent modeling approach overlooks the relationship between components and events. GAE-Log (Xie & Yang, 2023) combines components to build a knowledge graph. While it establishes the ownership relationship between components and events, it does not determine the progress of the component-based system. Therefore, we consider how to comprehensively model the relationship between components and events in order to fully capture the workflow of the system.

Secondly, significant changes in time intervals may be due to various performance issues, such as network latency or resource limitations (e.g., inadequate CPU, memory, or disk space). These issues can lead to the accumulation of events over time, which are then batch-written to the log. If we do not consider the timestamps of different events when analyzing the log sequence, we may overlook anomalies, which could result in system downtime and other issues. To solve this problem, LogNADS (Liu et al., 2021) processes timestamps, levels, dates, and other information in a vectorized manner, which can easily introduce additional noise that interferes with time features. SwissLog (Li, Chen, Jing, He, & Yu, 2023) conducts anomaly detection by combining time embedding and semantic embedding, but the resulting spliced features have a high dimensionality, leading to increased computation time and cost. Therefore, we consider introducing time features in heterogeneous graph analysis to capture both execution order and time anomalies simultaneously. However, in practical operation and maintenance, we have found that large-scale systems encompass a wide range of services and software, utilizing various programming languages and frameworks. This undoubtedly presents new challenges for anomaly detection tasks.

On the one hand, complex and diverse large-scale systems require a significant amount of time and energy to acquire labeled samples. Especially given a newly deployed system, it is more difficult to collect enough logs to train an anomaly detection model. In this case, LogTransfer (Chen et al., 2020) attempts to transfer learning methods, and TransLog (Guo et al., 2022) uses pre-training with adapter fine-tuning for cross system anomaly detection, but they still require some label data to adjust the transfer effect. On the other hand, the logs of different systems have their own expression styles. As shown in Fig. 1, the two abnormal log events from Linux and Windows systems, although they have similar log semantics, are quite different in syntax. Most word embedding methods, such as Word2vec (Mikolov, Chen, Corrado, & Dean, 2013), are highly susceptible to the syntax format in log templates. As a result, they struggle to extract reliable and comprehensive log semantic. DeepSyslog (Ding, Trajcevski, Scheuermann, Wang, & Keogh, 2022) uses smooth inverse frequency to further analyze the log context relationship. HitAnomaly (Huang et al., 2020) utilizes Transformer to encode log sequences. While the enhanced log semantics improve the anomaly detection performance of the model in the source system, it is difficult to effectively measure the similarity of cross-system logs due to the uneven semantics. Therefore, we aim to conduct cross-system semantic analysis without labeled data and develop a highly accurate and universal anomaly detection model.

To address the aforementioned issues, we propose LogGT, a novel cross-system log sequence anomaly detection method via heterogeneous graph feature and transfer learning. To comprehensively represent the relationship between components and log events, we design

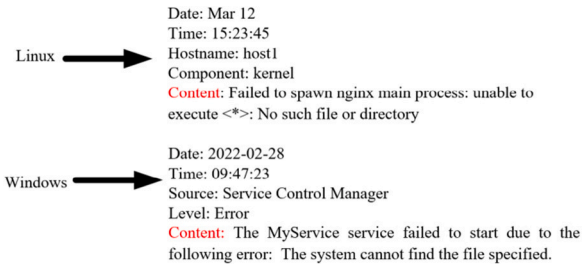


Fig. 1. Different system abnormal log events.

a heterogeneous graph, where log events are represented as nodes, and unique edges are established between log events from different components in the sequence. Then, to reduce the impact of syntax differences among various software systems and obtain more reliable node embeddings, LogGT utilizes the BERT model to extract log sentence vector. This vector is then inputted into Normalizing Flow for optimization, transforming it into a smoother Gaussian distribution. Next, we designed GTAT, which sets different type parameters to represent nodes and edges of different components to model heterogeneity. We do not need to manually determine the meta-paths of heterogeneous graphs, GTAT can capture the higher-order complex relationships between different types of nodes by determining the meta-relationships through message passing, while extracting the heterogeneous graph features by combining the time intervals. In addition, with the powerful representation capability of heterogeneous graphs, LogGT adopts a domain-adapted transfer learning technology. A semantic weighting method is designed to analyze the impact of semantics on feature transfer, and combine this weight with heterogeneous graph features to calculate the maximum mean difference between different systems. Then, this difference as a transfer loss to fine-tune the source domain model and achieve anomaly detection in the target domain system.

In summary, this paper makes the following contributions.

- We propose a heterogeneous graph modeling method to effectively combine different components and log events, and design a GTAT to further consider time intervals to capture the higher-order complex information of heterogeneous graphs.
- We employ an efficient log semantic representation method, which takes log sentences as BERT input and optimizes them using Normalizing Flow, resulting in graph node embeddings. Avoiding semantic confusion caused by syntax differences among different systems.
- We adopt a novel domain-adapted transfer learning technique and design a semantic weighting method, which calculate semantic weights and combine them with maximum mean difference loss to effectively transfer heterogeneous graph information in different software systems.

2. Related work

2.1. Log anomaly detection

The log anomaly detection process mainly includes three stages: log parsing, feature extraction, and anomaly detection. According to the general framework and our opted problem, we summarize the

Table 2
Comparison of log anomaly detection methods Sequence (S), Graph (G), Log Events (E), Time (T), Component (C), Parameter Value (P), Level (L), Adapter (A), Event Count Vector (EV), Event Index (EI), FASTText (FT), TF-IDF (TI), Word2vec (WV), Post-processing Algorithms (PA), Inverse Document Frequency (IF), Sentence-Bert (SB), Position Embedding (PE), Normalizing Flow (NF), Doc2vec (DV), Random Forests (RF), Transformer (TF), Bi-LSTM-Attention (BA), LSTM-Attention (LA), Transition Probability (TP), Transfer Learning (TL), Semantic Weight (SW). Smooth Inverse Frequency (SIF), Adversarial Domain Adaptation (ADA).

Methods	P1	P2	P3	P4
Xu, Huang, Fox, Patterson, and Jordan (2009)	EV	S, PCA	/	E
Ying et al. (2021)	TI	S, KNN	/	E
LogRobust (Zhang et al., 2019)	FT+TI	S, LA	/	E
Armand, Edouard, Piotr Bojanowski, Hervé, and Tomás (2016)	EV	S, SVM	/	E
LogEvent2vec (Wang et al., 2020)	EI+WV	S, RF	/	E
LightLog (Wang, Tian, Fang, Chen, & Qin, 2022)	WV+PA	S, TCN	/	E
LogBERT (Guo, Yuan, & Wu, 2021)	BERT	S, TF	/	E
Huang, Liu, Fung, Wang, and Yang (2023)	BERT	S,TF	/	E
LogEncoder (Qi et al., 2023)	BERT	S, LA	/	E
LayerLog (Zhang et al., 2023)	WV+TI	S, BA	/	E
MLog (Fu, Liang, & Xu, 2023)	BERT+ IDF	S, LSTM+CNN	/	E
DeepSysLog (Ding et al., 2022)	FT+SIF	S, LSTM	/	E+P
LogC (Yin et al., 2020)	EI	S, LSTM	/	E+C
SwissLog (Li et al., 2023)	BERT	S, BA	/	E+T
LogTAD (Han & Yuan, 2021)	WV	S, LSTM	ADA	E
LogTransfer (Chen et al., 2020)	Glove	S, LSTM	TL	E
TransLog (Guo et al., 2022)	SB	S, TF	A	E
GLAD-PAW (Wan, Liu, Wang, & Wen, 2021)	EI+PE	G, GAT	/	E
Log2vec (Liu et al., 2019)	TP+WV	G, DBSCAN	/	E
GAE-Log (Xie & Yang, 2023)	DV	G, GAE	/	E+C+L
Yan, Luo, and Shao (2023)	EI	G, GCN	/	E+T
LogGT	BERT NF	G, GTAT	SW+TL	E+C+T

existing research in a comparative Table 2 to focus on reviewing four key aspects: (1) P1: How to extract log features? (2) P2: What anomaly detection algorithms are utilized? (3) P3: Whether it supports cross-system analysis? (4) P4: What features are extracted from logs.

Log Parsing: The purpose of log parsing is to transform the original unstructured log messages into structured log templates, each of which represents a log event. LogMine generated log templates through grouping clusters. IPLoM (Makanju, Zincir-Heywood, & Milios, 2009) adopt an iterative partitioning strategy to divide log messages into groups. Ying et al. (2021) proposed a log parsing method based on N-gram and Frequent Pattern Mining (FPM) methods. According to Zhu et al. (2019) evaluation results, compared to other parsers, Drain He, Zhu, Zheng, and Lyu (2017) generates less noisy data and has higher detection accuracy. Therefore, we use Drain as the parser.

Feature extraction: Existing feature extraction methods can be mainly divided into two categories: event index-based methods and event semantic-based methods. Xu et al. (2009) used event count vector as input for the PCA to generate normal anomalous subspaces. LogEvent2vec (Wang et al., 2020)took the event index as input for Word2vec to obtain sequence features. Armand et al. (2016) used SVM for supervised training based on event count vector. This index-based approach does not provide any semantic information for the anomaly detection model, resulting in poor model robustness. For this reason, LogRobust (Zhang et al., 2019) used the FastText method to obtain log word vectors. LogTransfer (Chen et al., 2020) utilized Glove to extract template word vectors. LightLog (Wang et al., 2022) generated low-dimensional semantic features using Word2vec combined with PPA algorithm. These methods improve the robustness of the model by generating distributed log semantics. However, in different software systems with obvious syntactic differences and the phenomenon of "polysemy", traditional word embedding techniques are difficult to adapt to cross-system anomaly detection needs.

The BERT (Devlin, Chang, Lee, & Kristina, 2019) model proposed by the Google team has achieved promising results in NLP domains. It can dynamically generate vector by using a bidirectional Transformer structure to avoid ambiguity. In the log sequence anomaly detection task, LogBERT (Guo et al., 2021) used BERT to learn normal log sequence patterns. Huang et al. (2023) employed BERT to extract log semantics. For the convenience of cross-system log analysis, we also adopt the BERT model to take the entire log sentence as input instead of individual log words, obtain the log sentence vector, combines it with probabilistic modeling techniques to generate smooth and complete log semantics.

Anomaly Detection: With the advancement of artificial intelligence technology, log anomaly detection based on deep learning models has gained excellent results. LightLog (Wang et al., 2022) used a lightweight TCN model to detect abnormal log sequences. LogEncoder (Qi et al., 2023) combined LSTM and contrastive learning for anomaly detection. Layerlog (Zhang et al., 2023) used Bi-LSTM to model word-log-sequence hierarchical structure. MLog (Fu, Liang et al., 2023) used CNN to capture local log sequences and LSTM to capture global sequences representation. In response to existing methods that only consider log entries and ignore parameters in log events. DeepSyslog (Ding et al., 2022) encoded parameter vectors and used LSTM and MLP for anomaly detection. Although these methods have achieved high accuracy in the current system, it is difficult to replicate the same level of effectiveness when the model is transferred to other systems.

To achieve cross-system anomaly detection, LogTransfer (Chen et al., 2020) conducted transfer learning by sharing fully connected neural networks between the source and target domain systems; LogTAD (Han & Yuan, 2021) utilized adversarial domain adaptation techniques to make log data from different systems have similar distributions. TransLog (Guo et al., 2022) used pre-training and adapter fine-tuning to perform cross-system analyzing. However, they still require some labeled data from the target system for training, making it difficult to directly apply it to new systems lacking labeled information.

2.2. Heterogeneous graph neural network

Compared to traditional sequence structures, graph structures can be used to model and analyze complex data such as social networks and knowledge graphs. Heterogeneous graphs are able to express multiple entities and types of relationships, and are widely used in many fields. HAN (Wang et al., 2019) utilized heterogeneous graph attention network for graph node classification, sorting tasks. Fang et al. (2023) proposed a relation-aware graph convolutional network to deal with link prediction, entity alignment, and relation alignment. Hu, Dong, Wang, and Sun (2020) designed a heterogeneous graph Transformer to

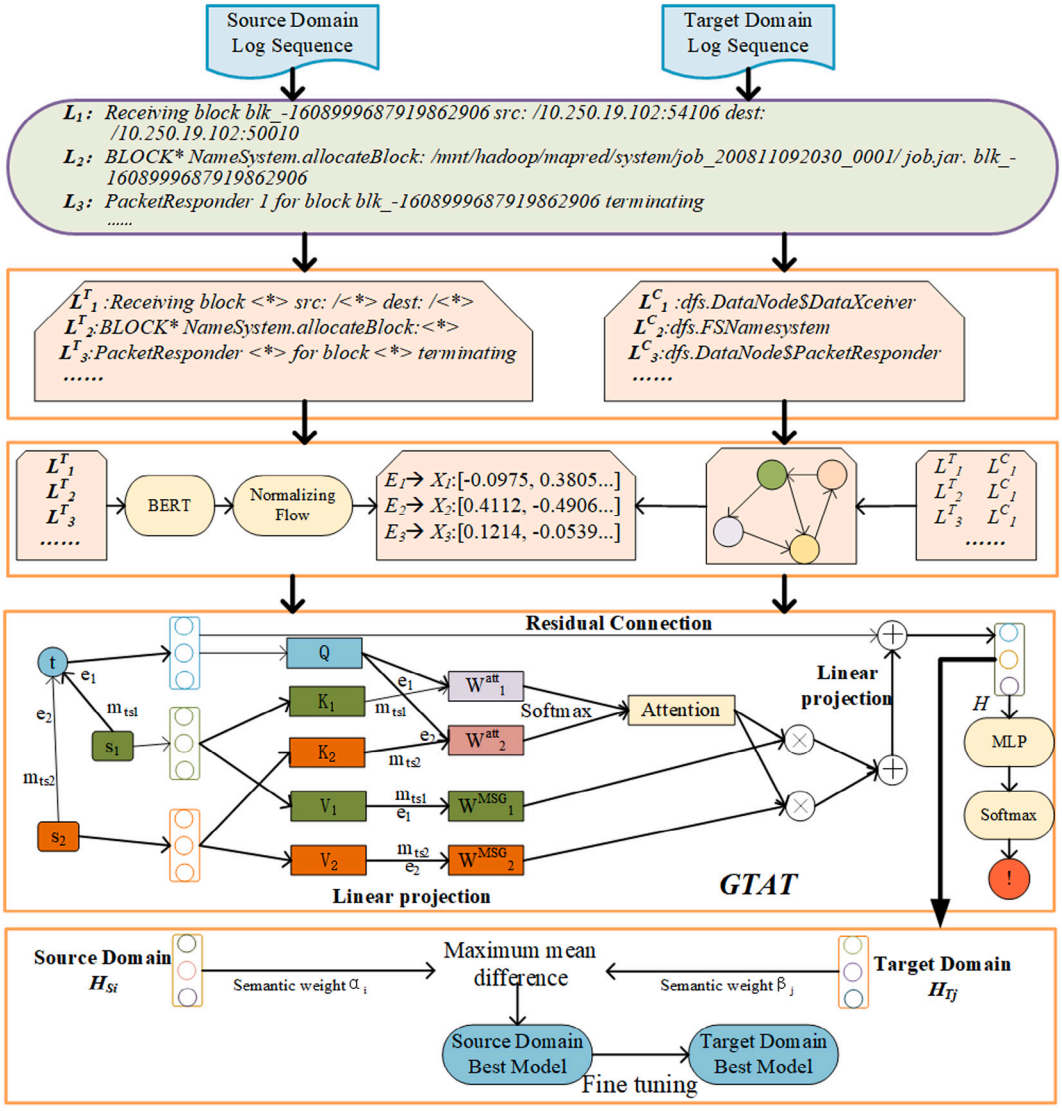


Fig. 2. The framework of LogGT.

handle dynamic graph over time. Chen et al. (2023) incorporate a graph contrastive learning paradigm to incorporate heterogeneous information into recommender systems. In order to fully capture the multilayer structure in multilayer heterogeneous graphs, Fu, Zheng, Huang, Yu, and Dong (2023) analyzed them level by level through decoupling, and enhance the model representation capability by combining contrastive learning.

In the field of log anomaly detection, GLAD-PAW (Wan et al., 2021) converted log sequences into session graphs and adds positional information to each node, using the readout function to generate graph representations. Log2vec (Liu et al., 2019) extracted multiple information from log messages, establishes heterogeneous graphs with multiple relationships coexisting, learns node embeddings using word2vec, and performed clustering-based anomaly detection using DBSCAN. GAE-Log (Xie & Yang, 2023) modeled log events and components into knowledge graphs and used adversarial training of autoencoders for anomaly detection. Yan et al. (2023) constructed a time-aware graph of logs, used graph convolutional networks to learn graph features, predict the next moment graph from the perspective of link prediction, and detect abnormal logs that deviate from the real graph structure. These methods convert log sequences into graph structures from different angles, and provide valuable references for log graph structure learning. However, there is still a lack of a comprehensive analysis of system component workflows and time interval variations in the graph.

3. Design of LogGT

3.1. Overview

To accurately detect abnormal log sequences of different software systems, we extract contents, components, and timestamp from the original log messages, and propose LogGT, whose overall structure is shown in Fig. 2. First, we take unstructured log messages from different systems (source domain, target domain) as input and parse them using the Drain algorithm to get structured log events (log templates). Second, LogGT builds a heterogeneous graph based on the parsed log events and components. In the heterogeneous graph, the same log event may come from the same component or different components. Next, in order to better analyze the semantic features of different systems, The BERT model combined with Normalizing Flow is employed to generate smooth graph node embeddings. Third, a GTAT is proposed to model heterogeneous graphs. It further detects time anomalies in log sequences by incorporating time interval into a multi-head attention mechanism. Finally, LogGT trains an optimal model in the source domain using cross-entropy loss. It then fine-tunes the model based on the maximum mean difference loss between graph features, and achieves cross-system anomaly detection tasks without the need for target domain label data.

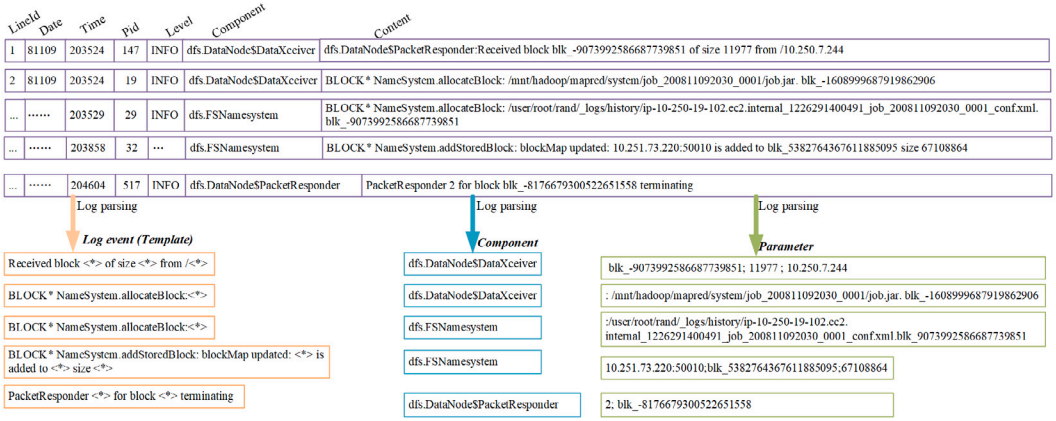


Fig. 3. The process of Log parsing.

3.2. Log parsing

Taking the HDFS dataset as an example, the log parsing process is shown in Fig. 3. We define a log as L, and we can get raw log message sequence $M = \{L_1, L_2, \ldots, L_n\}$, n represents that there are n logs in the sequence. Each message contains LineId, Date, Pid, and Content. To generate accurate log events LogGT uses the Drain algorithm to extract log template $L_i^T = \{w_1, w_2, \ldots, w_t\}$ from content, w_i represents ith word in a log event of length t. Each template describes the basic structure of the log and represents a specific operations during system operation, also called log events, where $\langle * \rangle$ represents the parameter values corresponding to the parsed variable. Next, LogGT further obtains component $L_i^C = \{c_1, c_2, \ldots, c_c\}$ and timestamp information $M = \{m_1, m_2, \ldots, m_n\}$. Where $c_i \in [1, c]$ represents ith word in the component with length c. Components represent modules in the system, such as software components, hardware devices, or network devices, and they record the source and location of logs. The timestamp records when the log was generated and is a visual reflection of system performance issues.

3.3. Heterogeneous graph construction

LogGT explores the logical relationships and interactions between components based on log sequences and proposes a heterogeneous graph construction method, as depicted in Fig. 4.

First, we convert each log sequence into a directed weighted heterogeneous graph $G = (P, E, A, R)$, Where P represents the set of nodes, E represents the set of edges, A represents the set of node types, and R represents the set of edge types. Specifically, the node set P consists of the different log events L^T in the log sequence. A represents the component to which each log event belongs, and we use the same color circle to represent the same type of log event in Fig. 4. The type mapping function for each node $p \in P$ is $\tau(p) : P \to A$. With this approach, LogGT introduces component information in the log sequence initially, further identifying the source of each log event. The set of edges E contains all the edges toward edge $e = (s, t, w_{st})$, where s represents source node, t represent target node, and which means that log event t occurs after s in the log sequence, w_{st} is the weight on the edge, its value is the number of times $L_s^T \to L_t^T$ appears in the log sequence. From this, it can be seen that the edges in heterogeneous graphs not only represent the execution order of all log events, but also further represent the statistical information of adjacent log events in the log sequence by setting weights. This statistical information is similar to the count vector of in the log sequence (Meng et al., 2019), which can effectively improve the accuracy of anomaly detection

In addition, the workflow of the system's components is important. Certain anomalies within the system can be easily detected within the component and may not be apparent in the sequence. For example,

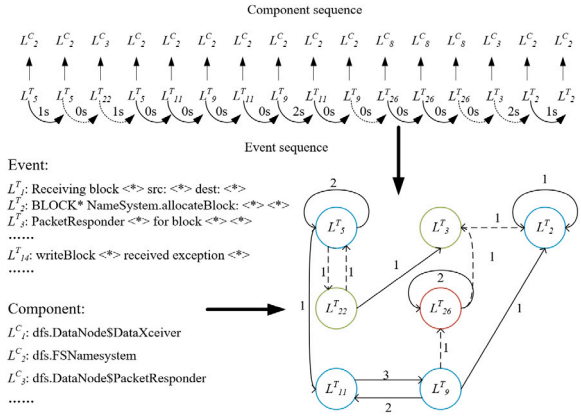


Fig. 4. Heterogeneous graph construction.

in a software system composed of multiple components, one of the components is responsible for network communication. The component responsible for communication is called the "communicator". It includes various log events, such as "receive packet" and "send packet". A common issue in this system is network blocking of the "communicator". This issue can only be detected within the component itself, as it needs to monitor its own send and receive queues for any backlog. However, there may only be vague warning information in the log sequence, making it difficult to identify the anomalies through log analysis. Therefore, LogGT further uses the component to establish edges of log events from the same components according to the sequential order relationship. R contains two types: 0 and 1, which represent the event interaction between the same components and different components, respectively. We use dashed lines and solid lines to represent them in Fig. 4, and its mapping function is $\phi(e) : E \to R$. Eventually, for an edge $e(s, t)$ linked from source node s to target node t, the traditional meta path can be further represented as this meta relationship $\langle \tau(s), \phi(e), \tau(t) \rangle$.

3.4. Heterogeneous graph node embedding

The embedding of nodes in the heterogeneous graph, which represents the semantic features of log events, serves as a comprehensive abstraction of system activities and plays a crucial role in the task of anomaly detection. For different system log data sets and anomaly detection goals, LogGT converts log events L_i^T into semantic vectors V_i. This conversion needs to meet the following requirements.

Reliability: The semantic vector should effectively represent log events with similar semantics and differentiate log events with distinguish semantics. Among them, log events with similar semantics

may come from different software systems. Additionally, the statements may be altered due to log updates or noise interference. For example, log event 1: "Received block ⟨∗⟩ of size ⟨∗⟩ from /⟨∗⟩", log event 2: "Received block ⟨∗⟩ from /⟨∗⟩", log event 3: "PacketResponder ⟨∗⟩ for block ⟨∗⟩ terminating". Where event 1 and event 2 have similar semantics, which means that the two semantic vectors generated by LogGT should have a higher cosine similarity. Events 1 and 3 represent different operations, so the cosine similarity between them should be low.

Integrity: The log semantic vector should consider the information of the log event as a whole, rather than just a single word vector. First, different words in a log event have varying effects on the semantic representation, and the complete log semantics should preserve this significance. For example, in the phrase "Adding an already existing block ⟨∗⟩", "block" is more effective in representing the log's semantics than the word "an". For two messages, "service interruption caused data loss" and "data loss caused service interruption" with different semantics, they have the same words but in different orders. If we ignore the word order, a misjudgment may occur.

To meet the aforementioned requirements, LogGT utilizes BERT combined with Normalizing Flow to generate embeddings for graph nodes. First, log events are inputted into the BERT model instead of individual words to generate log sentence vectors. The BERT model utilizes the encoder module of the Transformer for bidirectional encoding and has undergone extensive training on a large corpus. And most of the words in the log are written by programmers, including commonly used words for computer systems, which are included in the BERT corpus. Even if the log dataset contains out-of-vocabulary (OOV) words, the BERT model can still generate reasonable embeddings. This is due to its strong representational ability and the contextual information provided by the log. When generating the vector $V_i \in \mathbb{R}^{d_v}$ of template L_i^T, [CLS] is added as the beginning of the sentence.

$$V_i = BERT(CLS, L_i^T) \tag{1}$$

Next, to further optimize the generated sentence vectors for improved feature transfer, Normalizing Flow (Fu, Zheng et al., 2023) is used to transform the sentence vector V into a smooth isotropic Gaussian distribution X. Normalizing Flow is a generative model that performs feature changes by constructing a reversible transformation function. Formally, based on the sentence vector V, we first define the potential space Z and the observation space U of the log semantic features, and the samples z are sampled from the potential space Z, i.e. $z \sim p_z(z)$. The generation process of Normalizing Flow is given by:

$$u = f_{\Phi}(z) \tag{2}$$

where $f_{\Phi}(\cdot)$ is an invertible function, which consists of several differentiable transformations. According to the variable substitution theorem, if $f_{\Phi}(\cdot)$ is monotone differentiable and its derivative is not zero on the transformation interval, the probability density function of the observable space U is shown in:

$$p_U(u) = p_Z(f_{\Phi}^{-1}(u)) \left| \det \frac{\partial f_{\Phi}^{-1}(u)}{\partial u} \right| \tag{3}$$

where the observation space U is the sentence vector V, and $p_Z(z)$ conforms to the standard Gaussian distribution. The edge likelihood of U is maximized by performing a completely unsupervised approach in the potential space Z of semantic features, i.e., maximizing the probability density function of U.

$$\log p_U(u) = V, L^T \sim D\left(\log p_Z\left(f_{\Phi}^{-1}(u)\right) + \log \left| \det \frac{\partial f_{\Phi}^{-1}(u)}{\partial u} \right| \right) \tag{4}$$

where D denotes the dataset. During the training period, we fix the BERT model parameters and only optimizes the Flow parameters to finally learn an invertible function f_{Φ}^{-1}, which leads to a smooth node embedding X.

3.5. Graph transformer architecture with time intervals

Hu et al. (2020) utilized the Heterogeneous Graph Transformer (HGT) to effectively model heterogeneous graphs and achieved favorable outcomes in diverse downstream tasks. On this basis, LogGT further combines the time intervals of log events and designs a Graph Transformer Architecture with Time Intervals (GTAT). This allows for the detection of anomalies caused by changes in time intervals. We define the output of $l-1$ layer of node s as $H^{(l-1)}(s)$, which serves as the input of layer l. $H^{(0)} = X$ represents the initialized node feature, that is, the optimized log semantics. The entire GTAT process includes: Time Interval Matrix Calculation, Attention Score Calculation with Time Interval, Message Passing, Message Aggregation and Anomaly Detection.

3.5.1. Time interval matrix calculation

Performance issues often lead to changes in the time of events, such as network congestion that slows down the execution time of specific tasks, leading to delays in subsequent tasks and ultimately downtime. In this paper, we detect potential performance anomalies by mining the time intervals of logs and using them as time features. First, LogGT calculates the relative time interval m_{ij}^* between log events in the original log sequence based on the timestamp sequence $M = \{m_1, m_2, \ldots, m_n\}$. Subsequently, we calculate the mean μ and variance σ of all time intervals in the training set and perform standard deviation normalization:

$$m_{ij} = \frac{m_{ij}^* - \mu}{\sigma} \tag{5}$$

Based on the normalized value m_{ij}, we obtain the time interval sequence $M^1 = \{m_{21}, m_{32}, \ldots, m_{n(n-1)}\}$. Next, given a target node t and its neighbors $s \in N(t)$, $N(t)$ is all neighbor nodes of t, LogGT obtain the time feature

$$M^e = M^1 W_m^e + b_m^e \tag{6}$$

by using a linear layer to the time interval sequence M^1, where $W_m^e \in \mathbb{R}^{n \times d_v}$ and $b_m^e \in \mathbb{R}^{d_v}$ are the parameters and bias of the linear layer, $M^e = \{m_{21}^e, m_{32}^e, \ldots, m_{n(n-1)}^e\}$.

3.5.2. Attention score calculation with time interval

The source node s is projected onto the ith key vector $K^i(s)$ using linear projection $K\_linear_{\tau(s)}^i$. Similarly, project the target node t onto the ith query vector $Q^i(t)$ using linear projection $Q\_linear_{\tau(s)}^i$.

$$K^i(s) = K\_linear_{\tau(s)}^i(H^{(l-1)}(s) + m_{st}^e) \tag{7}$$

$$Q^i(t) = Q\_linear_{\tau(s)}^i(H^{(l-1)}(t)) \tag{8}$$

It is worth noting that in this process, LogGT combines time features m_{st}^e in the key vector to capture anomalies associated with time intervals. $\tau(s)$ represents the node type of s. By means of this indexing, nodes from different components in the heterogeneous graph are projected into their specific representation space. Following a linear projection, node vector from $\mathbb{R}^d$ to $\mathbb{R}^{\frac{d}{h}}$. h is the number of attention heads. $\frac{d}{h}$ is the vector dimension of each head. To ensure that the vector dimension of each head is uniform, keep d divisible by h so that $\frac{d}{h}$ is a positive integer. Then, to acquire the similarity between the query vector $Q^i(t)$ and the key vector $K^i(s)$, we employ a dot product operation to multiply the edge weight values w_{st} to obtain the result. For ith attention head:

$$ATT-head^i(s,e,t,w_{st}) = (K^i(s) W_{\phi(e)}^{ATT} Q^i(t)^T \cdot w_{st}) \cdot \frac{\mu\langle \tau(s), \phi(e), \tau(t) \rangle}{\sqrt{d}} \tag{9}$$

We distinguish different meta-relationships by adding a prior tensor $\mu\langle \tau(s), \phi(e), \tau(t) \rangle$, where $W_{\phi(e)}^{ATT} \in \mathbb{R}^{\frac{d}{h} \times \frac{d}{h}}$ represents an edge-based matrix reserved for different edge types $\phi(e)$. Next, h attention heads are connected to obtain the attention vector for each node pair. By

adding meta relationships, further differentiate the edges and nodes of different components to maximize their distribution differences. For each target point t, its attention scores

$$Attention(s,e,t) = \underset{\forall s \in N(t)}{Softmax}(\|_{i\in[1,h]} ATT-head^i(s,e,t,w_{st})) \tag{10}$$

from its neighbor $N(t)$ is collected, where $\mathrm{Softmax}(\cdot)$ is an activation function, and symbol "‖" represents feature stitching.

3.5.3. Message passing from source nodes to target nodes

The information is passed from the source node s to the target node t. In order to obtain the ith message headers $MSG-head^i(s,e,t)$, a linear layer $M\_linear^i_{\tau(s)}$ is used to combine time feature and project the source nodes of type $\tau(s)$ onto the ith message vector.

$$MSG\_head^i(s,e,t) = M\_linear^i_{\tau(s)}\left(H^{(l-1)}(s) + m^e_{st}\right) W^{MSG}_{\phi(e)} \tag{11}$$

where $W^{MSG}_{\phi(e)} \in \mathbb{R}^{\frac{d}{h}\times\frac{d}{h}}$ represents different types of edges. Then, the h message headers are connected to obtain the message $Message(s,e,t)$ for each node pair.

$$Message(s,e,t) = \underset{i\in[1,h]}{\|} MSG\_head^i(s,e,t) \tag{12}$$

Notably, we add meta-relations followed by message passing, which not only differentiates the distribution of nodes and edges, but also replaces the traditional method of manually determining meta-paths and facilitates cross-system heterogeneous graph feature transfer. Furthermore, by incorporating time features into the computation of attention scores and message transfer, the model can consider time interval and effectively capture the temporal correlation between nodes in the graph.

3.5.4. Message aggregation and anomaly detection

We aggregate the corresponding messages $Message(s,e,t)$ from source nodes s by using the multi-head attention score $Attention(s,e,t)$, resulting in an updated vector.

$$\tilde{H}^{(l)}(t) = \bigoplus_{\forall s\in N(t)} (Attention(s,e,t)\cdot Message(s,e,t)) \tag{13}$$

This allows us to aggregate the messages from all neighboring nodes $s \in N(t)$ with different feature distributions to the target node t, where $\oplus$ represents feature aggregation. The linear projection method $A\_linear_{\tau(t)}$ is used to process the vector $\tilde{H}^{(l)}(t)$, and the output $H^{(l)}(t)$ of the target node is obtained by using the residual connection method, which map target node back to a specific distribution of type $\tau(r)$.

$$H^{(l)}(t) = A\_linear_{\tau(t)}\left(\sigma(\tilde{H}^{(l)}(t))\right) + H^{l-1}(t) \tag{14}$$

Finally, we can get the node representation $H^{(L)}(t)$ by stacking L layers, and aggregate the features of all nodes to generate heterogeneous graph features H of the original sequence. The MLP with $\mathrm{Softmax}(\cdot)$ is used to output the detection result. As shown in Eq. (15). W_y represent training the weight matrix, b_y is bias terms.

$$\hat{y} = \mathrm{Softmax}(W_y \cdot H + b_y) \tag{15}$$

During the model training, LogGT uses the prediction outputs and the ground-truth to calculate the cross-entropy as the loss function.

3.6. Domain-adapted transfer learning

The heterogeneous graph proposed in this paper contains various relationships between components and events. It has more powerful representation capabilities than the original sequence. Experiments have shown that it exhibits better anomaly detection when conducting supervised learning. In order to share heterogeneous graph information from different systems, LogGT employs a novel domain-adapted transfer learning method to effectively apply the source domain model to the target domain dataset, even in the absence of labeled data in the target domain.

Specifically, Log events are processed by Normalizing Flow to obtain smooth node embeddings X, based on this, we design a semantic weighting calculation method. Firstly, LogGT calculates the correlation scores $score$ of the source domain semantic s_i and the target domain semantic t_i.

$$score_{s_i} = Sim\left(s_i, avg_T\right) \tag{16}$$

$$score_{t_j} = Sim\left(t_j, avg_S\right) \tag{17}$$

where $Sim(\cdot)$ denotes the calculation of cosine similarity, $avg_T = \frac{1}{m}\sum_{j=1}^{m} t_j$ and $avg_S = \frac{1}{n}\sum_{i=1}^{n} s_i$ are the average semantic features of all log sequences in the target domain and source domain, respectively. m and n represent the number of sequence in the target domain and source domain, respectively. Then, based on the correlation scores, the source domain attention weights α_i and the target domain attention weight β_j are obtained, respectively.

$$\alpha_i = \frac{\exp(score_{s_i})}{\sum_{k=1}^{n}\exp(score_{s_k})} \tag{18}$$

$$\beta_j = \frac{\exp(score_{t_j})}{\sum_{k=1}^{m}\exp(score_{t_k})} \tag{19}$$

Through this approach, LogGT analyzes the differences between the source and target domains from a semantic perspective. This makes it easier for the model to apply the semantic features learned from the source domain to the target domain, conduct preliminary semantic knowledge transfer, and avoid interference caused by syntax and structure. Subsequently, LogGT utilizes GTAT for feature extraction to obtain heterogeneous graph features H_{Si}, H_{Tj}. These features are the result of dynamic modeling and learning of nodes and edges in the graph, and they can be effectively applied to various downstream tasks. Next, by combining the weight value α_i, β_j, we obtain the weighted heterogeneous graph feature $\bar{H}_S$, $\bar{H}_T$.

$$\bar{H}_S = \frac{1}{n}\sum_{i=1}^{n}\alpha_i \cdot H_{Si} \tag{20}$$

$$\bar{H}_T = \frac{1}{m}\sum_{j=1}^{m}\beta_j \cdot H_{Tj} \tag{21}$$

Finally, LogGT calculates the feature differences between different systems, i.e. the maximum mean difference value. It then transfers the heterogeneous graph information from the source system S to the target system T in order to achieve domain-adapted transfer learning.

$$MMD[\mathcal{F},p,q] = \sup_{\|f\|_{\mathcal{H}}\le 1}\left(E_p[\bar{H}_S] - E_q[\bar{H}_T]\right) \tag{22}$$

where E_p and E_q represent the calculation expectation, sup represents finding the supremum, that is, the norm in the regenerative Hilbert space should be less than or equal to 1. Then, it is used as domain adaptation loss $MMD\_loss = MMD[\mathcal{F},p,q]$, to fine-tune the source domain model to obtain the target domain anomaly detection model. The entire process of Domain-adapted transfer learning as shown in Algorithm 1.

Similar to many other methods for detecting log anomalies, LogGT performs on-line system detection. During the training phase, we train the model using a complex source domain dataset. The maximum mean difference loss combining semantic and graph features is used to fine-tune source model, enabling it to adapt to the log sequences from the target domain. In the on-line detection process, the log sequence is parsed, a heterogeneous graph is constructed, nodes are embedded, and the trained GTAT model is used to obtain the representation of the heterogeneous graph. The predicted labels for this sequence are computed using the Softmax function, and are defined as either Normal or Abnormal.

Algorithm 1: Domain-adapted transfer learning Algorithm of LogGT

Input: Source domain: S_M, n_s: number of log sequence, n_t: number of log events. || Target domain: T_M, m_s: number of log sequence, m_t: number of log events. || N: number of epochs

Output: Target Domain Results: y^t

Source domain training:;
while $i < n_s 1$ **do**
  Heterogeneous graphs of source domain: $G^s = (P, E, A, R)$;
while $j < n_s 2$ **do**
  $V^s = BERT(CLS, L^T)$;
  $X^s = NormalizingFlow(V^s)$
for $epoch \leftarrow 1$ *to* N **do**
  $H_S = GTAT(G^s, X^s)$;
  $y^s_{pre} = \mathrm{softmax}(W \cdot H_s + b)$;
  $Loss = \frac{1}{n_s} \sum \left(-y^s_{true} \log y^s_{pre} \right)$
  Source Domain Optimal Model: $F^s(x)$
Target domain training:;
while $i < m_s 1$ **do**
  Heterogeneous graphs of target domain: $G^t = (P, E, A, R)$;
while $j < m_s 2$ **do**
  $V^t = BERT(CLS, L^T)$;
  $X^t = NormalizingFlow(V^t)$
Semantic weighting: Source domain $\rightarrow \alpha$; Target domain $\rightarrow \beta$;
for $epoch \leftarrow 1$ *to* N **do**
  $H_T = F^s(G^t, X^t)$;
  $\bar{H}_S = \frac{1}{n_s 1} \sum \alpha \cdot H_S$;
  $\bar{H}_T = \frac{1}{m_s 1} \sum \beta \cdot H_T$;
  maximum mean difference: $MMD\_loss(\bar{H}_S, \bar{H}_T)$;
  Obtain $F^t(x)$ by fine-tuning $F^s(x)$ based on MMD_loss;
New target domain data $\rightarrow x$;
return $y^t \leftarrow F^t(x)$;

3.7. Time complexity analysis

We now analyze the time complexity of LogGT to explore the efficiency of the algorithm. For GTAT, we stack l layers and use the self-attention mechanism of the h head for feature aggregation, the time complexity is $O(lhN_e^2 d)$, i.e. $O(N_e^2 d)$, N_e is the number of edges in G, d is the dimension of embedding. Compared to traditional sequence models such as RNN, its complexity is $O(Nd)$, N is the sequence length, $N > N_e$. But LSTM usually combines Attention to assign weights to different log events, and its time complexity further increases to $O(NN_l d)$, N_l is the number of log events. Experiments have shown that GTAT has better modeling performance than LSTM due to its neglect of component-related spatial structures. In addition, when performing anomaly detection in the target domain, feature transfer is achieved by calculating semantic weighted scores, with a time complexity of $O(N_1 N_2 d)$, N_2 is the number of log events in the target domain. Overall, LogGT achieves higher efficiency with lower time complexity.

4. Experiment

4.1. Experiment configuration

4.1.1. Dataset

To showcase LogGT's anomaly detection performance, we initially selected the publicly log datasets HDFS and BGL for comparative experiments (He, Zhu, He, & Lyu, 2020). Next, to perform cross-system log analysis, we introduced the Thunderbird dataset. We selected 3 datasets that can perform 6 sets of transfer learning analysis, which is sufficient to verify the cross-system anomaly detection effect of LogGT in different scenarios.

The reasons for choosing them are as follows: (1) Availability: All three datasets contain information such as components and timestamps during collection, which can fulfill the modeling requirements of existing log anomaly detection methods. (2) Complexity. They contain different numbers of templates and components, and the length of the templates varies. This can more comprehensively reflect the impact of the model. (3) Representative: They represent different software systems, respectively. Their logs have commonalities in semantics and individuality in syntax. In addition, these three datasets are commonly utilized in the field of log anomaly detection (Le & Zhang, 2022a).

Dataset Details As shown in Fig. 5, we extract one log from each dataset for detailed analysis.

In the HDFS dataset, the Date and Time are used to record the log time. Pid indicates the unique identifier of the process. The current log level is INFO, which is generated by the "dfs.DataNode$PacketResponder" component, and Content represents the original log message. HDFS is collected from 203 node clusters of AmazonEC2 platform and tagged by experts in the field of Hadoop. A total of 11,175,629 raw log messages are included, which are generated by 9 different components,including 46 templates. It records a unique block_id for each block operation and each identifier is uniquely labeled, whereby the dataset is partitioned into 575,061 sessions, including 558,223 normal sessions and 16,838 abnormal sessions.

Most of the fields in the BGL dataset are similar to HDFS. The label is abnormal if it is "1". RAS indicates the type of this log. BGL is collected from the BlueGene/L supercomputer system at Lawrence Livermore National Laboratory (LLNL) in Livermore, California, and contains a total of 4,747,963 raw log messages tagged by domain experts and generated by 14 different components, including 1848 templates. Among them, 348,460 logs were marked as abnormal. Unlike the HDFS data, BGL does not have an identifier recorded for each job execution, so we only use a sliding window to divide them into log sequences, and for each log sequence, if there are abnormal messages, the sequence is considered anomalous.

In the Thunderbird dataset, "an416" represents user information associated with the log event. "An416/an416" records the geographic or system location where the current event occurred. It was collected from the Thunderbird supercomputer system at the Albuquerque Sandia National Laboratory (SNL), which has 9024 processors and 27072 GB of memory, and is marked by experts in the field. "–" represents no exceptions, otherwise it is an abnormal log, consisting of a total of 3992351 logs generated by 54 different components, including 477 templates. Among them, 162953 logs are marked as abnormal. Similar to the BGL dataset, it also uses a sliding window to construct log sequences.

Data preprocessing: We initially organize the original log messages based on the "Time" and "Date" fields to prevent incorrect execution order due to noise interference. We then delete logs with empty "Component" or "Content" fields to avoid interference from invalid logs. Next, various "components" are extracted and assigned unique "IDs" for building the heterogeneous graphs. After parsing the logs, we discard the parameters (variables) part, and the new "event" field is used to replace the "content". Finally, we divide the logs according to the session window or sliding window to obtain the log sequence for training.

4.1.2. Baseline method

We compare LogGT with several different baseline methods. These baselines are as follows.

PCA (Xu et al., 2009): Using the log template index as input, the PCA algorithm is used to construct the normal and abnormal spaces of the template count vector.

HDFS

Date	Time	Pid	Level	Component	Content
81109	203524	47	INFO	dfs.DataNode$PacketResponder	dfs.DataNode$PacketResponder:Received block blk_-9073992586687739851 of size 11977 from /10.250.7.244

BGL

label	Timestamp	Date	Node	Type	Component	Level	Content
-	1117838570	2005.06.03	R02-M1-N0-C:J12-U11	RAS	Kernel	INFO	instruction cache parity error corrected

Thunderbird

label	Timestamp	Date	User	Location	Component	PID	Content
-	1131904443	2005.11.13	an416	An416/an416	crond	2260	synchronized to 10.100.16.250, stratum 3

Fig. 5. Logs from different datasets.

Table 3
Anomaly detection results in different datasets.

Datasets	HDFS			BGL		
Metrics	Precision	Recall	F1-score	Precision	Recall	F1-score
PCA	0.978	0.649	0.975	0.552	0.609	0.562
SVM	0.982	0.902	0.940	0.971	0.853	0.899
DeepLog	0.958	0.933	0.945	0.910	0.956	0.932
LogRobust	0.981	0.958	0.969	0.967	0.946	0.956
DeepSysLog	0.965	0.993	0.979	0.976	0.995	0.985
LogGT	0.986	0.981	0.983	0.996	0.993	0.995

SVM (Armand et al., 2016): A hyperplane is constructed to separate the high-dimensional space by using the template count vector and its labels as training instances. If the log sequence lies above the hyperplane, it is abnormal.

DeepLog (Du et al., 2017): A typical deep learning approach in the field of log anomaly detection, using a stacked LSTM network to learn normal log sequence patterns. The log sequences that deviate from the normal pattern are determined to be abnormal.

LogRobust (Zhang et al., 2019): FastText and TF-IDF algorithms are utilized to represent log templates as semantic vectors, while LSTM combined with attention is employed to identify abnormal log sequences.

LogTransfer (Chen et al., 2020): GloVe is used to capture semantic information from logs, while LSTM is employed to extract patterns from log sequences. Transfer learning methods are utilized to share a fully connected network between the source and target systems for cross-system anomaly detection.

DeepsysLog (Ding et al., 2022): An unsupervised sentence embedding method is utilized to extract log semantics, employing LSTM to capture log event context, and integrating event metadata(parameters) to obtain a comprehensive representation of log messages.

4.1.3. Evaluation metric

Log sequence anomaly detection as a typical classification problem, we adopt *Precision*, *Recall* and *F1-score* as evaluation metrics.

$$Precision = \frac{TP}{TP + FP} \tag{23}$$

$$Recall = \frac{TP}{TP + FN} \tag{24}$$

$$F1 - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{25}$$

where TP is the number correctly identified as abnormal, FP is the number incorrectly identified as abnormal, FN is the number of anomalies detected as normal.

4.1.4. Experimental environment

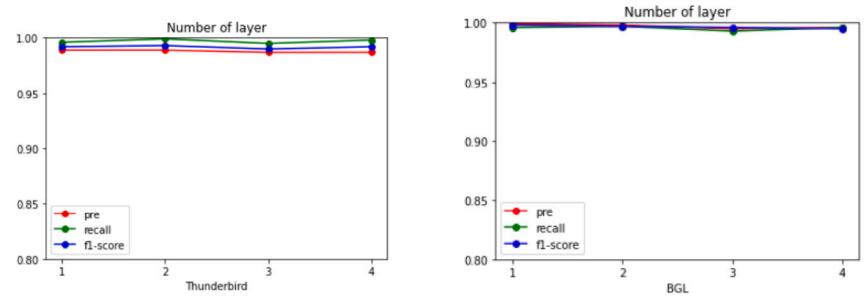
This experiment was conducted on a NVIDIA RTX 3090 24G GPU server, using Python 3.8, PyTorch 1.12.1, and CUDA 11.3. During the experiment, 80% of the dataset is selected as the training set, and the remaining 20% is allocated as the test set. We use an off-the-shelf service bert-as-service to obtain log sentence vectors, which use BERT as encoder and runs it as a service, the vector dimension is 768. Analyze heterogeneous graphs using GTAT, set the hidden layer dimension to 256. We stack $L = 2$ layer networks, and set the head number as $h = 4$. LogGT uses Adam (Kingma & Ba, 2015) as the optimizer, the batch size is 64. We train the model and stop training when the "loss" of the model stabilizes for multiple consecutive epochs.

4.2. Results and analysis

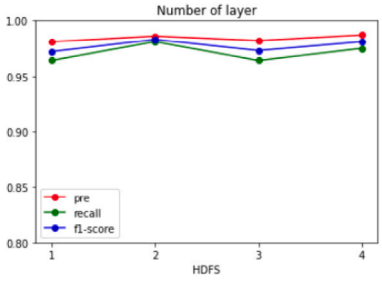
In order to preliminarily verify the effectiveness of the log semantic feature extraction method and anomaly detection model in LogGT, we conducted comparative experiments on the HDFS and BGL data sets without using transfer learning. The experimental results as shown in Table 3.

As we can see, LogGT achieves higher scores than other baseline models in the *F1-score* and *Precision*. *Recall* also exceeds 0.98. When comparing the two machine learning methods, the unsupervised learning PCA algorithm is inferior to SVM in all metrics. This is because PCA prioritizes the main aspects of the data and disregards the low-frequency information of abnormal samples. Particularly for the BGL dataset, which contains a variety of log templates, the performance of the PCA algorithm is further diminished. SVM obtained the second highest *Precision* after LogGT in the HDFS dataset, but its *Recall* was lower than that of other deep learning models. This is because SVM was not able to effectively learn the abnormal log features, causing the constructed classification hyperplane to misclassify some abnormal logs as normal. Comparing DeepLog and SVM, prediction effect of DeepLog had a significant improvement, which indicates that the deep learning model ave more advantages in the log sequence anomaly detection task. However, the metrics of DeepLog are lower than those of LogRobust for two reasons. First, DeepLog uses template index as input, which results in the loss of semantic information from logs. Second, LogRobust can effectively utilize supervised signals to distinguish between normal and abnormal log features. In addition, the *F1-score* of PCA, SVM, and DeepLog is low than LogRobust, DeepSysLog, and LogGT. This further indicates that index-based methods are easily affected by factors such as the evolution of log statements and noise, which can lead to misclassification or the overlooking of certain anomalies.

LogRobust's *Precision* in the HDFS dataset is slightly higher than DeepSysLog. However, its *F1-score* in two datasets is 1% and 2.9% lower than DeepSysLog, respectively. This is because DeepSysLog further analyzes the semantic relationships of the same log entries in different contexts when performing sentence embedding. Although DeepSysLog only utilizes normal log sequences for training, it further analyzed the metadata based on the log events, thereby achieving higher metrics. This proves that semi-supervised learning methods can improve anomaly detection performance by deeply analyzing log events and reasonably modeling log context information. LogGT improves *Recall* and *F1-score* by 7.9% and 4.3% respectively in the HDFS dataset compared to the SVM. In the BGL dataset, LogGT outperforms the LogRobust by 2.9%, 4.7%, and 3.9% in the three metrics, despite having more templates. One of the reasons is that LogGT utilizes the BERT model to acquire more dependable log sentence vectors and then

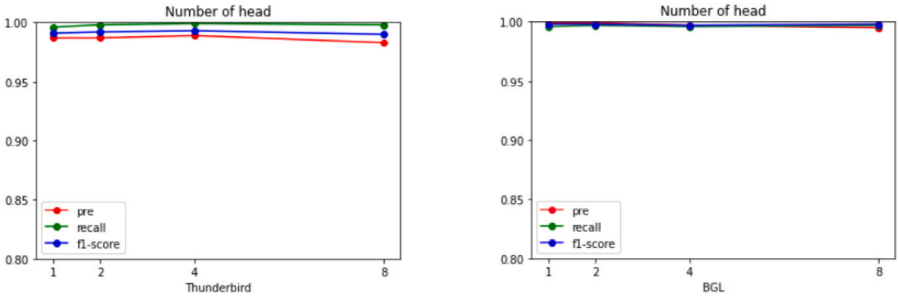


(a) Number of layer (Thunderbird) (b) Number of layer (BGL)

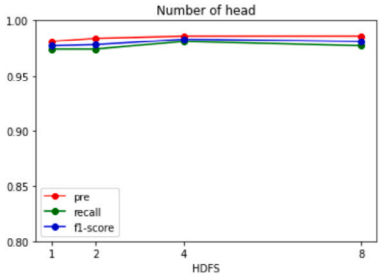


(c) Number of layer (HDFS)

Fig. 6. Anomaly detection results for different layer numbers.



(a) Number of head (Thunderbird) (b) Number of head (BGL)



(c) Number of head (HDFS)

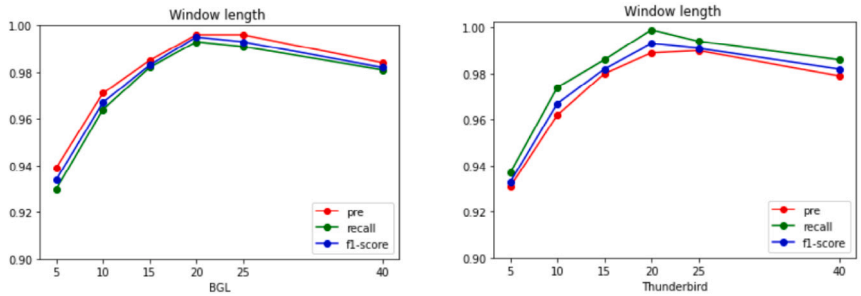
Fig. 7. Anomaly detection results for different head numbers.

optimizes them with Normalizing Flow to derive log semantics, making it adaptable to various complex systems. Compared to the optimal baseline model DeepSysLog, LogGT has only a slight decrease in *Recall*, but it is still very close to it. The reason is that some of the anomalies in log sequences may be contained in the metadata. Therefore, DeepSysLog is able to detect more anomalies. However, the other metrics of LogGT are better, indicating that we model heterogeneous graphs and use GTAT to analyze these graphs. This allows us to better distinguish between normal and abnormal log sequences.

4.3. Parameter sensitivity

We now conduct parameter sensitivity experiments to analyze LogGT. Important parameters include the number of attention heads h, the number of GTAT layers L, and the size of the sliding window.

As shown in Fig. 6, 7, in the Thunderbird dataset, the *Recall* is higher than the *Precision* and *F1-score*. This indicates that LogGT is able to detect almost all anomalies in this dataset. The anomaly detection is slightly improved when the head number is set to $h = 4$. The



(a) BGL (b) Thunderbird

Fig. 8. Anomaly detection results for different window size.

three metrics are more stable in the BGL dataset, all reaching 0.99, and the model is less affected by the number of heads and layers. In the HDFS dataset, the *Precision* is higher and the *Recall* is lower. This indicates that for LogGT misclassified some of the anomalies as normal, and the model achieved higher performance when the number of layers $L = 2$. The effect is basically close when the heads number $h = 8$ and $h = 4$. But with the increase of h and L the model has higher computational complexity and consumes more device resources, which is not conducive to the practical application of cross-system anomaly detection. Therefore, we ultimately set $h = 4$ to capture heterogeneous features from different angles and sets $L = 2$ to extract deeper heterogeneous features.

Firstly, we used sliding windows of equal size and step size to divide BGL and Thunderbird, ensuring that the model focuses on independent time periods within each window. At the same time, overlapping samples were not included in neighboring windows to prevent the artificial introduction of noise and to guarantee the reliability and continuity between log sequences.

Secondly, sliding windows of different sizes affect the sensitivity and accuracy of model. The smaller windows can capture the system state at a finer granularity, it is difficult to capture long-term anomalous behavior. As shown in Fig. 8, the sliding window changes from 5 to 40 with the trend of different metrics. When the window size is 5, the *F1-score* is the lowest. The reason for this is that a window that is too small contains limited information, and the model cannot effectively learn the operational state of the system. As the window increases, the sequential and temporal relationships of the log sequences can be fully exploited, and the best *F1-score* is achieved when the window size is set to 20. The effect decreases when the window size exceeds 25. This is because an excessively large window contains incomplete log behavior, which the model may misclassify as noise. Additionally, too large a window increases the complexity and computational cost of the model.

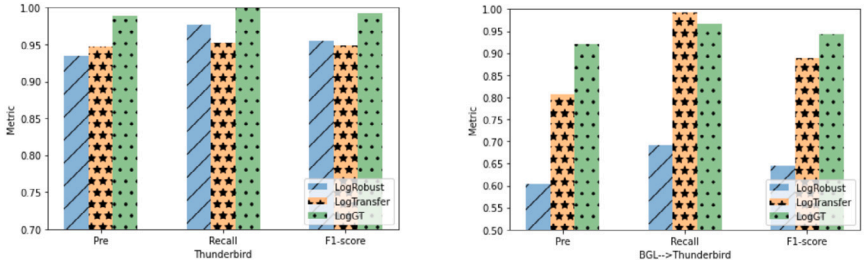
4.4. Analysis of transfer learning results

To evaluate the effectiveness of the LogGT cross-system log sequence anomaly detection method, we conduct further experiments using the Thunderbird dataset. We selected LogRobust, which, like LogGT, belongs to semantic-based methods, and LogTransfer, which utilizes transfer learning, as the comparison models to analyze the experimental results.

We first conducted an experimental analysis on the Thunderbird dataset. We then performed transfer learning to train the anomaly detection model using the BGL dataset as the source domain and the Thunderbird dataset as the target domain. This process is illustrated in Fig. 9. Comparing Fig. 9(a) and 9(b), it can be seen that the effect of the LogRobust varies widely. In scenario Fig. 9(b), the *F1-score* only reached 0.645, indicating that supervised learning models like LogRobust, after training on the source dataset, cannot be applied to other datasets and lack the ability to detect anomalies across systems. In LogTransfer, Glove is combined with global word co-occurrence and local context information of template words to obtain a cross-system log representation. Transfer learning is then performed by sharing a fully connected neural network. The *F1-score* of this method is 22.4% higher than that of LogRobust, which has shown effective cross-system log sequence anomaly detection. However, this method still requires retraining with labeled data from specific target domains. In contrast, LogGT adopts a domain-adapted transfer learning approach, achieving better results on the target domain dataset, with all metrics exceeding 0.9. Moreover, LogGT does not require labels from the target domain. It only calculates the maximum mean difference in an unsupervised manner to fine-tune the source system model.

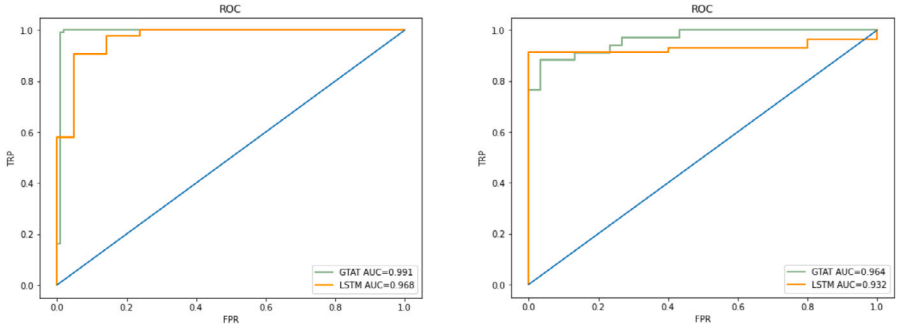
Subsequently, considering the differences and complexity within components and templates in different datasets, we successively extracted 25k, 40k, and 50k datasets from TB, BGL, and HDFS datasets to train the source domain model, which preserve the system's complete state as much as possible, while effectively highlighting the variations between different systems. The other two datasets are used as target domains for further analysis of the transfer learning effect. The results are shown in Table 4, where TB represents the Thunderbird dataset. It can be observed that when the BGL dataset is used as the source domain, transferring to other system datasets yields the best anomaly detection performance, with an *F1-score* exceeding 0.9. This is particularly true for BG→HDFS, where the *Precision* exceeds 0.96. The reason is that the HDFS dataset only contains 46 unique templates and 9 distinct components, with a single data type and low data complexity. By analyzing the semantic differences between BGL and HDFS using LogGT, better model transfer can be achieved. When the HDFS dataset is used as the source domain, the transfer learning effect is similar to that of the traditional unsupervised learning model PCA. This suggests that having sufficient anomaly knowledge in the source domain and utilizing multi-class templates are essential for ensuring the effectiveness of cross-system anomaly detection. Meanwhile, when TB is used as the source domain, which has the largest variety of components, the *Recall* is higher in the BGL dataset, which indicates that LogGT can detect most of the anomalies, but its *Precision* is lower because LogGT does not rely on the label information of the target domain at all, resulting in some normal templates in BGL being misclassified as anomalies. On the other hand, when TB is migrated to the HDFS dataset, all the metrics exceed 0.95. This is because TB contains more kinds of templates, and after fine-tuning, LogGT makes full use of TB's anomaly knowledge.

Assuming that the complexity of BGL, TB, and HDFS datasets is defined as complex, medium, and simple, respectively, according to the template and component types. The results in Table 4 demonstrate that through the semantic weighting method proposed in this paper, LogGT effectively utilizes the heterogeneous graph feature of source system. This ensures the transfer effect from complex systems to other systems and optimizes the effect from medium to complex systems by calculating the and maximum mean difference of feature across different systems. Furthermore, LogGT outperforms unsupervised machine learning algorithms in transferring knowledge from simple to complex systems, even without utilizing the target system labels.



(a) Thunderbird (b) BGL→Thunderbird

Fig. 9. Cross-system anomaly detection effect.



(a) ROC curve in Thunderbird (b) ROC curve in BGL

Fig. 10. ROC curve analysis of different models.

Table 4
Anomaly detection results of different target systems.

Metrics	Precision	Recall	F1-score
TB→BGL	0.668	0.992	0.798
TB→HDFS	0.957	0.998	0.978
BGL→TB	0.871	0.973	0.919
BGL→HDFS	0.970	0.955	0.962
HDFS→TB	0.607	0.685	0.644
HDFS→BGL	0.928	0.512	0.660

4.5. Analysis of heterogeneous graph modeling effect

To validate the efficacy of the heterogeneous graph modeling approach in this paper, we further extracted component information from the original log sequence and merged it with the log template through concatenation. We also use BERT to extract sentence vectors, Normalizing Flow is used to optimize embedding, and then use LSTM for anomaly detection. We call the method as "LSTM", and compare LSTM with GTAT. Specifically, we select Thunderbird datasets with a greater variety of component types and BGL datasets with significant differences in normal and abnormal samples, and extracted 8k and 10k data from Thunderbird and BGL, respectively for comparative experiments. Considering the uneven distribution of log templates in components and the imbalanced characteristics of normal and abnormal samples, the AUC is selected as the evaluation indicator in this section to better validate the effectiveness of the model.

Define (1) $FRP = \frac{FP}{TN+FP}$. The proportion of all normal log sequences that are incorrectly detected by the model as abnormal log sequences. (2) $TPR = Recall$. The ROC curve is the line connecting all the coordinate points (FPR, TPR) obtained by changing various thresholds (normal and abnormal log discrimination criteria) with FPR as the horizontal coordinate and TPR as the vertical coordinate.

The details are shown in Fig. 10. In the Thunderbird dataset, the normal and abnormal samples are balanced, but there are more component categories, and the log templates are unevenly distributed among the components. For instance, the component 'kernel' contains about half of the templates, while the component 'crond' only has four templates. From Fig. 10(a), it can be seen that GTAT has a higher AUC value and better anomaly detection performance. This indicates that the method of constructing heterogeneous graphs in LogGT can better model the relationship between components and templates, thereby improving anomaly detection performance. In the extracted BGL dataset, the percentage of abnormal log sequences is about 6%, and it can be seen from Fig. 10(b) that the AUC value of our method is 3.2% higher than that of LSTM in the unbalanced dataset. The issue is that while the template features are enhanced to some extent by the component and template splicing, it is not effective in detecting component-related anomalies. This results in the mis-classification of certain abnormalities. On the other hand, LogGT's heterogeneous graph includes diverse relationships between components and templates. The GTAT is used for modeling a heterogeneous graph and can detect multiple abnormal log sequences.

5. Conclusion

In this paper, we propose LogGT, a cross-system log anomaly detection method via heterogeneous graph feature and transfer learning. LogGT introduces component and time information on the basis of log events and innovatively models log sequences as heterogeneous graphs. In order to address the issue of similar semantics but significant syntactic differences among different systems, the BERT is employed to extract log sentence vectors, and Normalizing Flow algorithm is used optimize them. Considering time interval anomalies, a GTAT network is proposed to extract heterogeneous graph features. Besides, we design a semantic weighted calculation method to better transfer graph features from the source system to the target system. Comparing the five baseline models on real-world datasets, LogGT achieves the highest Pre and F1-score. The analysis of transfer learning effect reveal that LogGT outperformed the other methods by at least 8.6% in F1-score. In addition, the AUC value of our heterogeneous graph exceeds 0.96 in different datasets, which outperforms the sequence model.

CRediT authorship contribution statement

Peipeng Wang: Conceptualization, Methodology, Validation, Investigation, Writing – original draft, Writing – review & editing. **Xiuguo Zhang:** Conceptualization, Methodology, Supervision, Funding acquisition. **Zhiying Cao:** Investigation, Supervision, Funding acquisition. **Weigang Xu:** Methodology, Validation. **Wangwang Li:** Methodology, Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (Grant No. 52231014) and Liaoning Province Applied Basic Research Program Project, China (Grant No. 2023JH2/101300195).

References

Armand, J., Edouard, G., Piotr Bojanowski, M. D., Hervé, J., & Tomás, M. (2016). Fasttext.zip: Compressing text classification models. CoRR abs/1612.03651.

Chen, M., Huang, C., Xia, L., Wei, W., Xu, Y., & Luo, R. (2023). Heterogeneous graph contrastive learning for recommendation. In *Proceedings of the sixteenth ACM international conference on web search and data mining* (pp. 544–552). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3539597.3570484.

Chen, R., Zhang, S., Li, D., Zhang, Y., Guo, F., Meng, W., et al. (2020). Logtransfer: Cross-system log anomaly detection for software systems with transfer learning. In *2020 IEEE 31st international symposium on software reliability engineering*. http://dx.doi.org/10.1109/ISSRE5003.2020.00013.

Devlin, J., Chang, M.-W., Lee, K., & Kristina, T. (2019). 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* (pp. 4171–4186). http://dx.doi.org/10.18653/V1/N19-1423.

Ding, H., Trajcevski, G., Scheuermann, P., Wang, X., & Keogh, E. (2022). Deepsyslog: Deep anomaly detection on syslog using sentence embedding and metadata. *IEEE Transactions on Information Forensics and Security*, *17*, 3051–3061. http://dx.doi.org/10.1109/TIFS.2022.3201379.

Du, M., Li, F., Zheng, G., & Srikumar, V. (2017). Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security* (pp. 1285–1298). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3133956.3134015.

Fang, Y., Li, X., Ye, R., Tan, X., Zhao, P., & Wang, M. (2023). Relation-aware graph convolutional networks for multi-relational network alignment. *ACM Transactions on Intelligent Systems and Technology*, *14*(2), http://dx.doi.org/10.1145/3579827.

Fu, Y., Liang, K., & Xu, J. (2023). Mlog: Mogrifier lstm-based log anomaly detection approach using semantic representation. *IEEE Transactions on Services Computing*, *16*(5), 3537–3549. http://dx.doi.org/10.1109/TSC.2023.3289488.

Fu, C., Zheng, G., Huang, C., Yu, Y., & Dong, J. (2023). Multiplex heterogeneous graph neural network with behavior pattern modeling. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining* (pp. 482–494). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3580305.3599441.

Guo, H., Lin, X., Yang, J., Zhuang, Y., Bai, J., Zheng, T., et al. (2022). Translog: A unified transformer-based framework for log anomaly detectiong. https://arxiv.org/abs/2201.00016.

Guo, H., Yuan, S., & Wu, X. (2021). Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks* (pp. 1–8). http://dx.doi.org/10.1109/IJCNN52387.2021.9534113.

Han, X., & Yuan, S. (2021). Unsupervised cross-system log anomaly detection via domain adaptation. In *Proceedings of the 30th ACM international conference on information & knowledge management* (pp. 3068–3072). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3459637.3482209.

He, S., He, P., Chen, Z., Yang, T., Su, Y., & Lyu, M. R. (2021). A survey on automated log analysis for reliability engineering. *ACM Computing Surveys*, *54*(6), http://dx.doi.org/10.1145/3460345.

He, S., Zhu, J., He, P., & Lyu, M. R. (2020). Loghub: A large collection of system log datasets towards automated log analytics. CoRR abs/2008.06448.

He, P., Zhu, J., Zheng, Z., & Lyu, M. R. (2017). Drain: An online log parsing approach with fixed depth tree. In *2017 IEEE international conference on web services*. http://dx.doi.org/10.1109/ICWS.2017.13.

Hu, Z., Dong, Y., Wang, K., & Sun, Y. (2020). Heterogeneous graph transformer. In *The web conference* (pp. 2704–2710). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3366423.3380027.

Huang, S., Liu, Y., Fung, C., He, R., Zhao, Y., Yang, H., et al. (2020). Hitanomaly: Hierarchical transformers for anomaly detection in system log. *IEEE Transactions on Network and Service Management*, *17*(4), 2064–2076. http://dx.doi.org/10.1109/TNSM.2020.3034647.

Huang, S., Liu, Y., Fung, C., Wang, H., & Yang, H. (2023). Improving log-based anomaly detection by pre-training hierarchical transformers. *IEEE Transactions on Computers*, *72*(9), 2656–2667. http://dx.doi.org/10.1109/TC.2023.3257518.

Kingma, D. P., & Ba, J. (2015). Adam: A method for stochastic optimization. In *3rd international conference on learning representations*.

Le, V.-H., & Zhang, H. (2022a). Log-based anomaly detection with deep learning: How far are we? In *Proceedings of the 44th international conference on software engineering* (pp. 1356–1367). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3510003.3510155.

Le, V.-H., & Zhang, H. (2022b). Log-based anomaly detection without log parsing. In *Proceedings of the 36th IEEE/ACM international conference on automated software engineering* (pp. 492–504). IEEE press, http://dx.doi.org/10.1109/ASE51524.2021.9678773.

Li, X., Chen, P., Jing, L., He, Z., & Yu, G. (2023). Swisslog: Robust anomaly detection and localization for interleaved unstructured logs. *IEEE Transactions on Dependable and Secure Computing*, *20*(4), 2762–2780. http://dx.doi.org/10.1109/TDSC.2022.3162857.

Liu, X., Liu, W., Di, X., Li, J., Cai, B., Ren, W., et al. (2021). Lognads: Network anomaly detection scheme based on log semantics representation. *Future Generation Computer Systems*, *124*, 390–405. http://dx.doi.org/10.1016/j.future.2021.05.024.

Liu, F., Wen, Y., Zhang, D., Jiang, X., Xing, X., & Meng, D. (2019). Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security* (pp. 1777–1794). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3319535.3363224.

Makanju, A. A., Zincir-Heywood, A. N., & Milios, E. E. (2009). Clustering event logs using iterative partitioning. In *Proceedings of the 15th ACM SIGKDD international conference on knowledge discovery and data mining* (pp. 1255–1264). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/1557019.1557154.

Meng, W., Liu, Y., Zhu, Y., Zhang, S., Pei, D., Liu, Y., et al. (2019). Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *Proceedings of the twenty-eighth international joint conference on artificial intelligence* (pp. 4739–4745).

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. In *1st international conference on learning representations*.

Qi, J., Luan, Z., Huang, S., Fung, C., Yang, H., Li, H., et al. (2023). Logencoder: Log-based contrastive representation learning for anomaly detection. *IEEE Transactions on Network and Service Management*, *20*(2), 1378–1391. http://dx.doi.org/10.1109/TNSM.2023.3239522.

Wan, Y., Liu, Y., Wang, D., & Wen, Y. (2021). Glad-paw: Graph-based log anomaly detection by position aware weighted graph attention network. In *Pacific-Asia conference on knowledge discovery and data mining* (pp. 482–494). Cham, Switzerland: Springer, http://dx.doi.org/10.1007/978-3-030-75762-5_6.

Wang, X., Ji, H., Shi, C., Wang, B., Ye, Y., Cui, P., et al. (2019). *Heterogeneous graph attention network*. New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3308558.3313562.

Wang, J., Tang, Y., He, S., Zhao, C., Sharma, P. K., Alfarraj, O., et al. (2020). Logevent2vec: Logevent-to-vector based anomaly detection for large-scale logs in Internet of Things. *Sensors*, *20*(9), http://dx.doi.org/10.3390/s20092451.

Wang, Z., Tian, J., Fang, H., Chen, L., & Qin, J. (2022). Lightlog: A lightweight temporal convolutional network for log anomaly detection on the edge. *Computer Networks*, *203*, http://dx.doi.org/10.1016/j.comnet.2021.108616.

Xie, Y., & Yang, K. (2023). Log anomaly detection by adversarial autoencoders with graph feature fusion. *IEEE Transactions on Reliability*, http://dx.doi.org/10.1109/TR.2023.3305376.

Xu, W., Huang, L., Fox, A., Patterson, D., & Jordan, M. I. (2009). Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on operating systems principles* (pp. 117–132). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/1629575.1629587.

Yan, L., Luo, C., & Shao, R. (2023). Discrete log anomaly detection: A novel time-aware graph-based link prediction approach. *Information Sciences*, *647*, http://dx.doi.org/10.1016/j.ins.2023.119576.

Yin, K., Yan, M., Xu, L., Xu, Z., Li, Z., Yang, D., et al. (2020). Improving log-based anomaly detection with component-aware analysis. In *2020 IEEE international conference on software maintenance and evolution* (pp. 667–671). http://dx.doi.org/10.1109/ICSME46990.2020.00069.

Ying, S., Wang, B., Wang, L., Li, Q., Zhao, Y., Shang, J., et al. (2021). An improved knn-based efficient log anomaly detection method with automatically labeled samples. *ACM Transactions on Knowledge Discovery from Data*, *15*(3), http://dx.doi.org/10.1145/3441448.

Zhang, C., Wang, X., Zhang, H., Zhang, H., andZhang, Jiahua., Liu, C., et al. (2023). Layerlog: Log sequence anomaly detection based on hierarchical semantics. *Applied Soft Computing*, *132*(109860), http://dx.doi.org/10.1016/j.asoc.2022.109860.

Zhang, X., Xu, Y., Lin, Q., Qiao, B., Zhang, H., Dang, Y., et al. (2019). 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* (pp. 807–817). New york, NY, USA: Association for computing machinery, http://dx.doi.org/10.1145/3338906.3338931.

Zhu, J., He, S., Liu, J., He, P., Xie, Q., Zheng, Z., et al. (2019). Tools and benchmarks for automated log parsing. In *Proceedings of the 41st international conference on software engineering: software engineering in practice* (pp. 121–130). IEEE press, http://dx.doi.org/10.1109/ICSE-SEIP.2019.00021.