Anomaly Detection on Interleaved Log Data With Semantic Association Mining on Log-Entity Graph

Guojun Chu, Jingyu Wang, *Senior Member, IEEE*, Qi Qi, *Senior Member, IEEE*, Haifeng Sun, *Senior Member, IEEE*, Zirui Zhuang, *Member, IEEE*, Bo He, *Member, IEEE*, Yuhan Jing, Lei Zhang, and Jianxin Liao, *Senior Member, IEEE*

***Abstract*—Logs record crucial information about runtime status of software system, which can be utilized for anomaly detection and fault diagnosis. However, techniques struggle to perform effectively when dealing with interleaved logs and entities that influence each other. Although manually specifying a grouping field for each dataset can handle the single grouping scenario, the problems of multiple and heterogeneous grouping still remain unsolved. To break through these limitations, we first design a log semantic association mining approach to convert log sequences into Log-Entity Graph, and then propose a novel log anomaly detection model named Lograph. The semantic association can be utilized to implicitly group the logs and sort out complex dependencies between entities, which have been overlooked in existing literature. Also, a Heterogeneous Graph Attention Network is utilized to effectively capture anomalous patterns of both logs and entities, where Log-Entity Graph serves as a data management and feature engineering module. We evaluate our model on real-world log datasets, comparing with nine baseline models. The experimental results demonstrate that Lograph can improve the accuracy of anomaly detection, especially on the datasets where entity relationships are intricate and grouping strategies are not applicable.**

***Index Terms*—Log analysis, anomaly detection, content association, knowledge discovery, heterogeneous graph.**

Received 2 June 2024; revised 24 December 2024; accepted 30 December 2024. Date of publication 13 January 2025; date of current version 13 February 2025. This work was supported in part by the National Key R&D Program of China under Grant 2024YFE0200800; in part by the National Natural Science Foundation of China under Grant 62321001, Grant 62471055, Grant 62406039, Grant 62401080, Grant U23B2001, Grant 62101064, Grant 62171057, Grant 62201072, Grant 62001054, and Grant 62071067; in part by the Ministry of Education and China Mobile Joint Fund under Grant MCM20200202 and Grant MCM20180101; and in part by the Fundamental Research Funds for the Central Universities under Grant 2024PTB-004. Recommended for acceptance by P. Pelliccione. *(Guojun Chu and Jingyu Wang contributed equally to this work.) (Corresponding authors: Haifeng Sun; Bo He.)*

Guojun Chu, Jingyu Wang, Qi Qi, Haifeng Sun, Zirui Zhuang, Bo He, Yuhan Jing, and Jianxin Liao are with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing 100083, China (e-mail: chuguojun@bupt.edu.cn; wangjingyu@bupt.edu.cn; qiqi8266@bupt.edu.cn; hfsun@bupt.edu.cn; zhuangzirui@bupt.edu.cn; hebo@bupt.edu.cn; jingyuhan@bupt.edu.cn; liaojx@bupt.edu.cn).

Lei Zhang is with China Unicom Network Communications Company Ltd., Beijing 100032, China (e-mail: zhangl83@chinaunicom.cn).

Digital Object Identifier 10.1109/TSE.2025.3527856

I. Introduction

In recent years, the scale and complexity of software systems are rapidly increasing. Meanwhile, it is a crucial issue to ensure the stability and reliability of large-scale systems. Logs record the detailed runtime information of the system, which is instrumental in monitoring system status, diagnosing faults and analyzing user behaviors [1], [2], [3], [4]. However, logs are massive and unstructured data generated all the time [5], which makes it impossible for engineers to manually inspect each log message. To address this practical issue, a large number of studies focusing on automated log analysis tools and anomaly detection models are emerging [6].

Log-based anomaly detection is one of the most important tasks in AIOps (Artificial Intelligence for IT Operations). A temporary failure of a large-scale software system may result in a loss of thousands of dollars [7]. To minimize such losses, DeepLog [8] and LogRobust [9] are two representative log anomaly detection models in the past few years, which leverage event template sequences and semantic vectors to detect anomalous patterns in log sequence, respectively. Recent work such as [10], [11], [12] adopts word embedding techniques and sequence models (e.g., Transformers [13]) for supervised anomaly detection. In addition, large language model such as ChatGPT [14] can be applied to this task, but its performance seems to be unremarkable [15].

Interleaving Problem. Due to the concurrency of events within the system, there will be log messages from different sources generated at the same second and collected in arbitrary order [16]. The log messages related to the same entity (e.g., a block, server or user) are not guaranteed to be consecutive in the sequence. An example is shown in Fig. 1, where a number of irrelevant logs are inserted between log message $L3$ and $L11$ related to block $blk\_15708201$. The interleaved logs can interfere with traditional anomaly detection methods, resulting in incorrectly analysis of the patterns of log sequences. Even for graph-based methods which leverage event transition graph [17] to detect anomalies, the interleaving problem can lead to inaccurate estimation of transition probabilities, as well as inappropriate edges appeared in the graph.

It is pointed out in [18] that **grouping strategy** can effectively improve the performance of log anomaly detection models. That is, group the logs according to a specific field such

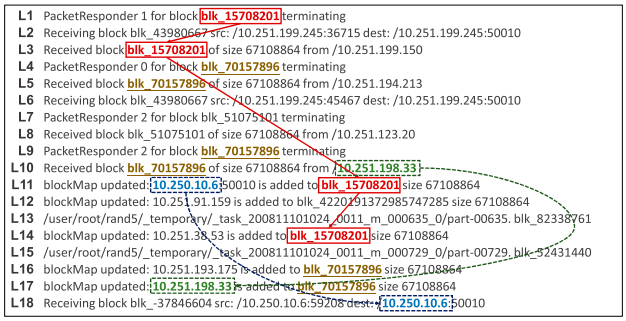


Fig. 1. An example of interleaving problem and semantic association between non-consecutive logs.

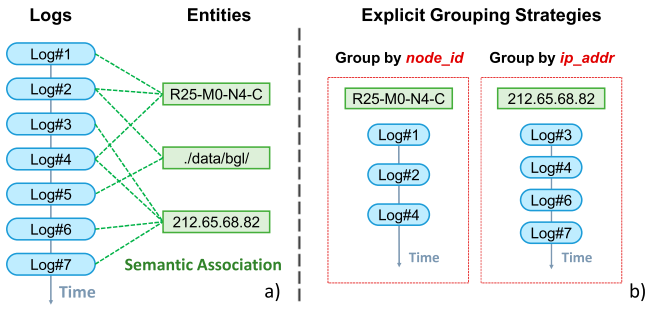


Fig. 2. An illustration of how the intrinsic semantic association serves as an alternative to explicit grouping strategies. The left part shows an example of Log-Entity Graph, in which the blue rounded rectangle represents a log node, while the green rectangle represents an entity node. The blue arrows and the green dashed lines represent the temporal association and semantic association, respectively. The right part shows two log sequences grouped by different fields.

as *block_id*, *node_id*, *user_id*, to obtain a complete and consecutive event records about an entity. For example, LogAssist [19] explicitly groups logs into event sequences to assist practitioners with log analysis. Due to the exclusion of irrelevant log messages, traditional methods are able to achieve excellent performance. However, grouping strategy has the following limitations in practice:

1) Engineers have to manually specify the field for grouping. Since the log format varies for different systems, inspecting logs to select a proper field will require additional manual labor, and will reduce the generalizability of the model.
2) A system may have **multiple** grouping strategies. If the logs record the interaction between several users and nodes, existing methods such as [8], [20] have to choose only one grouping field at a time (i.e., either $user\_id$ or $node\_id$) while ignoring the other one. It is unable to take into account both the patterns of user behavior and the patterns of machine running state.
3) Some logs may not contain any field that can be used for grouping. In other words, the grouping field does not cover the full set of logs. Besides, the grouping strategy may be less effective if the groups are **heterogeneous**, which means that there is significant variability in the event template distribution of different groups.

Motivated by the issues above, we hope to find an "implicit" grouping solution that can automatically discover the **semantic association** between logs and entities, while also handling complex cases of multiple and heterogeneous groupings. Here, semantic association refers to the fact that multiple logs are directly related to one or more common entity in terms of content. For example, in Fig. 1, four log messages $L1$, $L3$, $L11$, $L14$ are related to the same *block_id*, while another two log messages $L11$ and $L18$ are related to the same *ip_address*. Such additional knowledge can be utilized as an alternative to grouping strategy, and can further discover more complex anomalies (e.g. multiple events influencing each other). Another example is shown in Fig. 2. In the left part of this figure, we present a novel structure named Log-Entity Graph, which can be utilized to manage the log data, and exploit intrinsic semantic association. There are three possible grouping fields in this example, but explicit grouping strategy has to select only one grouping field at a time. If raw log messages are grouped by *node_id*, then $Log\#2$ will be adjacent to $Log\#4$ in the sequence, and not adjacent to $Log\#5$. If we use semantic association to implicitly group the logs, then $Log\#2$ and $Log\#4$ will be indirectly connected through the node $R25\text{-}M0\text{-}N4\text{-}C$. Meanwhile, $Log\#2$ and $Log\#5$ will also be connected through the ip address $212.65.68.82$.

However, existing anomaly detection methods have overlooked this semantic association, and only focused on sequential relationships within grouped log messages. In related works [9], [21], [22], fields carrying grouping information (e.g., *block_id* and *ip_address*) are often treated as variable parameters and removed from log messages during preprocessing [23]. As a result, existing models are unable to take advantage of those inherent relationships. To summarize, the contribution of mining log semantic association for capturing anomalous patterns has been underestimated. Especially in industrial settings, large-scale software systems are often heterogeneous, which makes grouping strategies unavailable.

In this paper, we propose **Lograph**, a novel log analysis and anomaly detection model based on Log-Entity Graph. We first design an effective data mining approach to extract entities from log messages. Then, we construct a heterogeneous graph named **Log-Entity Graph** to establish the log semantic association. Utilizing the knowledge embedded in the Log-Entity Graph can improve the accuracy of log-based anomaly detection without grouping. The performance of Lograph is evaluated on real-world log datasets, compared with nine anomaly detection models. The experimental results demonstrate that our model can improve the average accuracy by 2%-8% on non-grouped datasets.

The main contributions of this paper are as follows:

- A novel log-based anomaly detection model **Lograph** is proposed, which is capable of handling interleaved log data, as well as complex scenarios where multiple logs and entities influence each other.
- A log semantic association mining approach and a heterogeneous graph structure named **Log-Entity Graph** is proposed to discover implicit relationships. To the best of our knowledge, we are the first to exploit such association.

- Extensive experiments are conducted on real-world log datasets. The evaluation results demonstrate the superiority of the proposed model, especially when grouping strategies are not available.

The rest of this paper is organized as follows: Section II presents background and motivation of this work. Then, we describe the design of our log anomaly detection model in detail in Section III. The experimental results and discussions on necessary issues are presented in Section IV and Section V, respectively. Related work is introduced in Section VI. Finally, we conclude our work in Section VII.

II. Background and Preliminaries

A. Problem Statement

The log sequence can be regarded as a series of events that occur continuously in the system. Given N log messages $\mathbf{L} = \{L_1, L_2, \cdots, L_N\}$ sorted in chronological order, and a fixed window size H, the whole log dataset $\mathbf{L}$ can be cut into several log sequences $\mathbf{L}_k$ with length of H, where $\mathbf{L}_k = \{L_{k+1}, L_{k+2}, \cdots, L_{k+H}\}$ and $k \in [0, N-H]$. For each log sequence $\mathbf{L}_k$, a label $Y_k \in \{0, 1\}$ indicating whether $\mathbf{L}_k$ is related to an anomalous state of the system is assigned to it. The goal of log-based anomaly detection is to predict the value of Y_k based on the log messages that appear in the log sequence $\mathbf{L}_k$ or earlier.

B. Heterogeneous Graph

A heterogeneous graph is a special graph structure containing multiple types of nodes or multiple types of edges. Formally, it is defined as follows:

Definition 1: **A Heterogeneous Graph** is defined as a graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ in which each node $v \in \mathcal{V}$ and each edge $e \in \mathcal{E}$ are associated with their mapping functions $\psi(v) : \mathcal{V} \to \mathcal{A}$ and $\xi(e)$: $\mathcal{E} \to \mathcal{R}$, respectively. $\mathcal{A}$ and $\mathcal{R}$ denote the sets of predefined node types and edge types, where $|\mathcal{A}| + |\mathcal{R}| > 2$.

Definition 2: **A Meta-Path** in heterogeneous graph is defined as a path $\phi = v_1 \xrightarrow{R_1} v_2 \xrightarrow{R_2} \cdots \xrightarrow{R_m} v_{m+1}$, describing a relation $R = R_1 \circ R_2 \circ \cdots \circ R_m$ between v_1 and v_{m+1}, where $\circ$ denotes the composition operator on relations.

In a heterogeneous graph, the path connecting two nodes may contain edges with different types, which are called "meta-path". The nodes connected via a meta-path are "meta-path based neighbors". For example, the graph in Fig. 2 contains two types of nodes and two types of edges, where three possible meta-paths are involved in: *log-entity-log*, *entity-log-entity* and *log-log-log* paths.

C. Conventional Solutions

Conventional log-based anomaly detection solutions can be divided into the following two categories:

Sequence-based model first extracts template words from logs, and uses word embedding models [24] to generate a semantic representation for each log message. After that, sequence models such as Bi-LSTM [9] or Transformer [11] are leveraged to exploit temporal association and to predict subsequent event templates. However, the structural information and semantic association are ignored during preprocessing and semantic representation.

Graph-based model first parses the logs into event templates, and builds an event transition graph in which each node corresponds to an event, and each edge reflects the probability of events occurring one after the other [16], [25]. For example, if an event template $E2$ frequently (85%) appears immediately after $E1$ in the log sequence, then an edge $E1 \to E2$ with a weight of 0.85 will be connected in the graph. Although these models adopt Graph Neural Networks (GNNs) to detect anomalous nodes and anomalous edges [26], [27], their graph construction actually focus on only temporal order of logs.

Also, various training schemes are adopted, each with their own application scenario. For example, supervised model [9] trains a classifier to assign anomaly scores for each sample. Unsupervised model predicts the subsequent top-K event templates [21], or seeks a hypersphere to isolate anomalous samples using SVDD [28].

D. How Interleaving Problem Affects Conventional Solutions

In this subsection, we further elaborate on the implications of **interleaving problem** with the following example. Suppose that the normal event template sequence of an entity is $E1 \to E2 \to E3 \to E4$. If the logs of three different entities arrive at the same time, the following event template sequence can be obtained after sorting the log messages by timestamp: $E1_a \to E2_a \to E1_b \to E3_a \to E2_b \to E1_c \to E4_a \to E2_c \to E3_c \to E3_b \to E4_b \to E4_c$, where a, b, c are three entities. As the number of entities and the length of log sequences increase, there are more possibilities for the permutations of event templates. The diversity makes it difficult for sequence-based methods to learn patterns of normal sequences, with entity information ignored. For instance, an anomalous pattern with missing event templates is $E1 \to E4$ cannot be identified because it appears in the normal log sequences. Existing graph-based methods are slightly less affected since they can capture the differences in transition probabilities as well as paths between normal and anomalous sequences.

Conventional solutions can use grouping strategies (e.g., group by *username*) to split the above event sequence into three short sequences of length 4. However, if both $E2$ and $E3$ are related to another entity (e.g., a *server*) in the system, the historical running status $E2 \to E3 \to E2 \to E2 \to E3 \to E3$ of this server cannot be obtained. Such interleaving status is suitable to be represented by a graph, which integrates all the grouping possibilities. Conventional grouping strategies only take into account one path in the graph at a time, and cannot handle multiple paths simultaneously. The cases above inspire us to construct Log-Entity Graph to manage the logs.

III. Methodology

A. Log Preprocessing

Since logs are semi-structured text, they are usually parsed into event templates before anomaly detection using log

TABLE I
EXAMPLES OF THE TOKEN NAMES AND TYPES

Type	Token Name	Type	Token Name
Timestamp	1134160405	English Word	synchronized
IP Address	10.251.75.228	Block ID	blk_3587508140
Date	2022-10-07	Time	15:26:59
Node ID	R25-M0-N4-C	Hexadecimal	0x11e7ed80
Directory	./data/bgl/	User Name	tbird-admin1
Method	workerEnv.init()	Component	dfs.DataNode

parsers [29], [30]. For example, the log message "Receiving block $blk\_78652903$ src: $/10.251.29.239{:}47816$ dest: $/10.251.29.239{:}50010$" can be parsed into a log template "Receiving block <*> src: <*> dest: <*>", where each wildcard "<*>" represents a variable parameter. However, log parsing ignores the variable parameters and only reserves the event template. Although the content of logs can be understood by human with all the template words, the ignored variable parameters also carry some relational information [23]. In the example above, the event template describes a data receiving event, but it is not clear where the data comes from. To guide the establishment of the semantic association between logs, we first extract all the "entities" described in each log message, which is one of the unique features of this work. Then, we follow the procedure in [11] to directly remove non-character tokens for log parsing. Eventually, each log message is preprocessed into an event template and a set of entities.

Entity Extraction. Entities are usually identifiers with specific formats, indicating a *server*, a *block*, etc. Since it is impractical to extract all the entities based on manually designed regular expressions, we propose an automated entity extraction approach through character-level statistical analysis. Table I lists the names and types of some tokens that often appear in the log data. Intuitively, entities should have strong randomness in content and weak randomness in occurrence. That is, entities tend to be a mixture of letters, digits and symbols, and may appear frequently in a short time. To model this, we define the following two statistical indices:

(1) $TokenComplexity\ (TC)$ is the number of consecutive segments of letters, digits, or symbols in the token.

(2) $RecurrenceFrequency\ (RF)$ is the number of occurrences of the token in a sliding window.

For example, a block entity "blk_3587508140" can be divided into three consecutive parts: "blk", "_", "3587508140", and thus its TC value is 3. If a block entity appears 20 times in a month (or other time span), then its RF value is 20. If both TC and RF values are larger than their respective thresholds, the token will be recognized as an entity. We also find that using *case-sensitive* calculation for TC significantly improves recall. This is a simple yet effective method to extract the entities automatically. We will further evaluate and discuss the possibility of the omission and redundancy of extracted entities in Section IV-D and Section V.

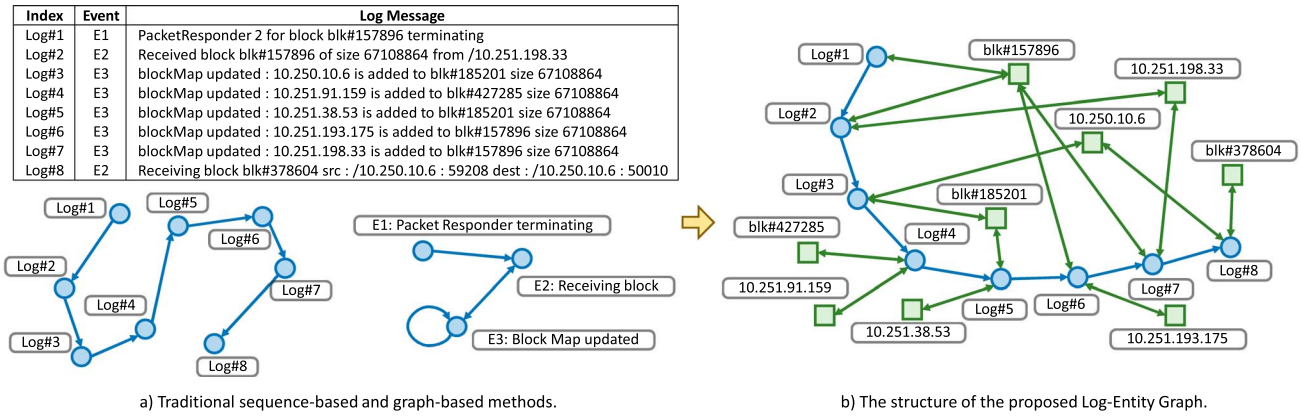
B. Log-Entity Graph Construction

In this subsection, we describe how to construct a Log-Entity Graph in detail, which is also the main difference between the proposed method and existing ones. Log-Entity Graph contains two types of nodes (i.e., log and entity), as well as two types of edges (i.e., "log-to-log" and "log-to-entity") in the graph. For each log message, we first connect it with all the entities mentioned in its content via "log-to-entity" edges, focusing on the semantic association. Then, we connect each log with the adjacent logs in chronological order via "log-to-log" edges, focusing on the temporal association. For example, the log message "Received block $blk\_35875126$ of size 67108864 from $/10.251.110.68$" contains two entities: "$blk\_35875126$" and "$/10.251.110.68$". This log node is connected with two entity nodes and two adjacent logs in the whole sequence through four edges. Since a log message may contain no entities, it is necessary to ensure the connectivity of the graph through "log-to-log" edges.

Fig. 3 illustrates an example of the construction of Log-Entity Graph. For a given log sequence containing eight log messages, the constructed Log-Entity Graph consists of 8 log nodes (denoted by blue circles) and 9 entity nodes (denoted by green squares). Specifically, the entity "blk#157896" is connected with four log nodes (Log#1, Log#2, Log#6 and Log#7), because it appears in the content of the corresponding four log messages. Also, Log#3 and Log#8 establish a semantic association through a common entity "10.250.10.6". By comparison, sequence-based anomaly detection methods such as [9], [10], [11] can be modeled as connecting log nodes into a chain, and exploiting temporal dependencies of neighboring nodes on the chain, as shown in the bottom left of Fig. 3(a). That is, a sequence-based method can be regarded as a degenerated version obtained by removing all the entity nodes from Log-Entity Graph. For conventional graph-based methods such as [16], [25], [31], an event transition graph is constructed as shown in the bottom right of Fig. 3(a). The edges record the transition probabilities between adjacent event templates, which also only reflect temporal (or sequential) relationship of the log messages.

Graph Snapshot Mechanism. An entity can be associated with thousands of log messages, which makes the graph extremely large in scale. To reduce the graph scale and accelerate model training, we adopt a graph snapshot mechanism. Since anomaly detection is expected to be real-time, new logs are streaming in every second, while logs from a long time ago may have little impact on current running state of the system. We therefore maintain a dynamic graph in which both logs and entities have their own "*expiration date*". That is, new nodes are continuously inserted into the graph, while expired nodes (i.e., historical logs outside the sliding window) are removed from the graph. We also assume that entities which have not appeared for more than a time interval τ (e.g., 60 minutes) are no longer relevant to the current anomaly, and are considered as different entities. The purpose of setting an expiration date is not only to reduce the computational overhead, but also to avoid being misled by anomalies that have already been fixed long ago.

Snapshots of the dynamic graph are collected as training and testing samples for our model. Formally, for a given log sequence $\mathbf{L}_k = \{L_{k+1}, L_{k+2}, \cdots, L_{k+H}\}$ and corresponding



Index	Event	Log Message
Log#1	E1	PacketResponder 2 for block blk#157896 terminating
Log#2	E2	Received block blk#157896 of size 67108864 from /10.251.198.33
Log#3	E3	blockMap updated : 10.250.10.6 is added to blk#185201 size 67108864
Log#4	E3	blockMap updated : 10.251.91.159 is added to blk#427285 size 67108864
Log#5	E3	blockMap updated : 10.251.38.53 is added to blk#185201 size 67108864
Log#6	E3	blockMap updated : 10.251.193.175 is added to blk#157896 size 67108864
Log#7	E3	blockMap updated : 10.251.198.33 is added to blk#157896 size 67108864
Log#8	E2	Receiving block blk#378604 src : /10.250.10.6 : 59208 dest : /10.250.10.6 : 50010

a) Traditional sequence-based and graph-based methods. b) The structure of the proposed Log-Entity Graph.

Fig. 3. The comparison between different methods in terms of graph construction.

manual label Y_k, we collect all the log neighbors and entity neighbors related to each log message in $\mathbf{L}_k$ on the dynamic graph. In this process, the log nodes that appear before the first log L_{k+1} in the sequence but have not yet expired will be excluded. Then, a sub-graph $\mathcal{G}_k$ (i.e., a snapshot) is constructed based on the collected neighbor nodes, and the label Y_k will be assigned to $\mathcal{G}_k$. The number of nodes in a sub-graph is approximately #logs $\times$ (1 + avg. #entities per log). Based on our statistical results on public datasets, each log contains an average of 3 to 4 entities. That is, if the sequence to be detected contains 20 logs, then the number of nodes in the sub-graph will be 80 to 100. Thus, the scale of the graph used for training and testing can be controlled within an acceptable range.

Algorithm 1 describes the process of graph construction, where a dictionary $\mathcal{D}$ records the latest log message related to a given entity. In this algorithm, line#7 to line#18 describe how to establish semantic association, and line#19 to line#21 describe how to establish temporal association.

Algorithm 1 Construction of Log-Entity Graph

Require: Log messages $\mathbf{L}$, and a maximum time interval τ.
Ensure: A Log-Entity Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$.

1: Initialize the dictionary $\mathcal{D} \leftarrow \emptyset$ for recording the latest log related to each given entity
2: Initialize the node set $\mathcal{V} \leftarrow \emptyset$ and the edge set $\mathcal{E} \leftarrow \emptyset$
3: **for** log message $L_k \in \mathbf{L}$ with index k **do**
4: Extract all the entities $\mathbf{E}(L_k)$ from log message L_k
5: Extract timestamp $T(L_k)$ from log message L_k
6: Insert a new log node L_k into the node set $\mathcal{V}$
7: **for** entity $E \in \mathbf{E}(L_k)$ **do** ▷ Semantic Association
8: **if** entity E exists in dictionary $\mathcal{D}$ **then**
9: Obtain the latest log $L_j \leftarrow \mathcal{D}(E)$ related to E
10: **if** time interval $T(L_k) - T(L_j) \leq \tau$ **then**
11: Add bidirectional edges L_kE, EL_k into $\mathcal{E}$
12: Update the latest log $\mathcal{D}(E) \leftarrow L_k$
13: **end if**
14: **else**
15: Insert a new entity node E into the node set $\mathcal{V}$
16: Insert $\mathcal{D}(E) \leftarrow L_k$ into the dictionary
17: **end if**
18: **end for**
19: **if** $k \geq 1$ **then** ▷ Temporal Association
20: Add a unidirectional edge $L_{k-1}L_k$ into $\mathcal{E}$
21: **end if**
22: **end for**

C. Semantic Representation

Log Semantics. We employ a log representation procedure that has been proven to be effective in existing work [9], [10], [22], which mainly consists of three steps. The first step is *Tokenization*. Logs usually contain non-character tokens (e.g., delimiters and punctuations), as well as composite words written in Camel Case [32]. In this step, all the non-character tokens are removed, and composite words are split into individual words according to Camel Case. The second step is *Vectorization*. In this step, the semantic vectors of each word are obtained through a pre-trained word2vec model such as FastText, Glove and BERT [33], [34], [35]. Following the existing work [22], we adopt a pre-trained Glove model to generate word vectors with length of d=300. The third step is *Aggregation*. The semantics of log messages are composed of the semantics of the words, but the importance of each word varies. In this step, we aggregate the word vectors in a given log message L based on TF-IDF [36] algorithm:

$$\mathbf{v}(L) = \sum_{w \in L} \mathrm{TF}(w, L) \cdot \mathrm{IDF}(w) \cdot \mathbf{v}(w) \tag{1}$$

where $\mathbf{v}(w)$,$\mathbf{v}(L)$ represents the word vector of w and the sentence vector of L, respectively. TF (Term Frequency) reflects the frequency of each word in a single log message, while IDF (Inverse Document Frequency) reflects the rarity of each word in the whole dataset. Here, IDF is pre-calculated based on the logs in training set. If an out-of-vocabulary word appears in the testing set, its IDF will be equal to the IDF of the word that appears only once.

Entity Semantics. Without the guidance of domain knowledge, it is difficult to obtain the natural language semantics of the entity itself. As an alternative, we aggregate the sentence vectors of related logs to indirectly represent the semantics of

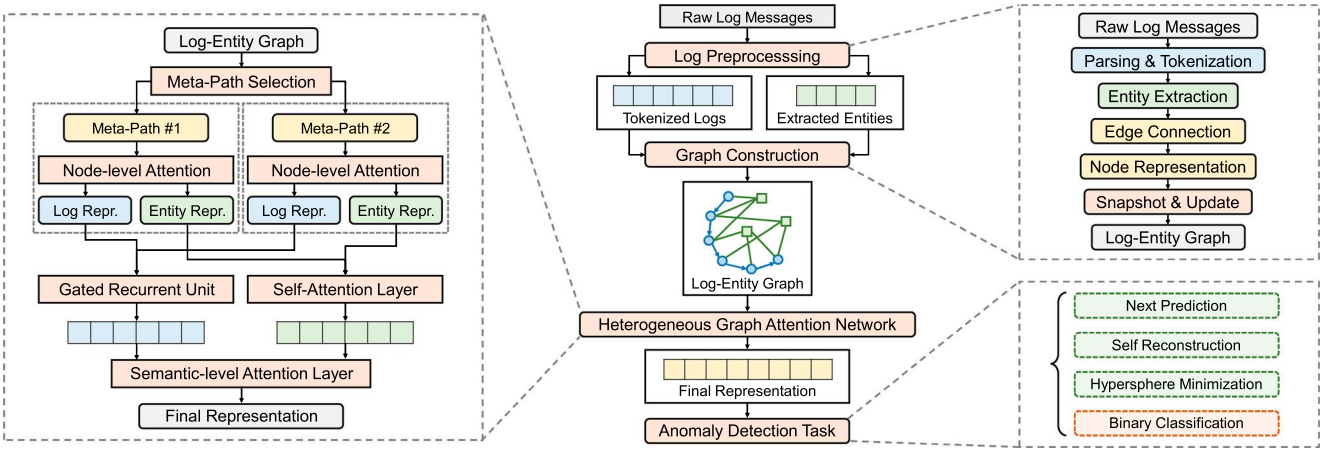


Fig. 4. An overview of the structure of the proposed anomaly detection model.

the entity. For each entity E appeared in a log sequence $\mathbf{L}_k$, we collect the latest H=20 historical log messages that contains E in content, denoted by $\mathbf{L}(E)$. That is, any log message L in $\mathbf{L}(E)$ should satisfy the following two conditions: 1) The entity E appears in the content of L. 2) The timestamp of L is not larger than that of the last log appeared in $\mathbf{L}_k$. After that, we treat an event template as a *word*, and treat a sequence of logs related to an entity as a *sentence*. Based on the definitions above, a *document* is composed of a set of sentences describing entities, and TF-IDF algorithm can be used again to aggregate the semantic vectors of each log in $\mathbf{L}(E)$:

$$\mathbf{v}(E) = \sum_{L \in \mathbf{L}(E)} \text{TF}(L, E) \cdot \text{IDF}(L) \cdot \mathbf{v}(L) \tag{2}$$

where $\mathbf{v}(E)$ represents the semantic vector of entity E, and $\mathbf{v}(L)$ represents the sentence vector of log L, which can be calculated by formula (1). Here, TF actually reflects the frequency of each event template appeared in a log sequence related to a given entity, while IDF reflects the proportion of entities that is related to a specific event template from the perspective of the entire entity set.

D. Heterogeneous Graph Attention Network

Heterogeneous Graph Attention Network [37] adopts a hierarchical attention structure. First, a node-level attention layer is used to aggregate the meta-path based neighbors. Then, a semantic-level attention layer is used to aggregate different meta-paths. The overview of our model is illustrated in Fig. 4.

Meta-Path Selection. We adopt three meta-paths that can reveal the relationship between different nodes in our model. For each entity node, its entity neighbors are collected via *entity-log-entity* meta-paths in the graph. For each log node, its log neighbors are collected via both *log-entity-log* and *log-log-log* meta-paths in the graph. Using more meta-paths do not improve the accuracy significantly, but slow down the training process of the model.

Node-Level Attention. For a given node, its neighbors in a specific meta-path ϕ are of different importance. The node-level attention layer mines the relationship between different nodes in the heterogeneous graph, and each neighbor node is assigned an attention score to reflect the importance. It is pointed out in [37] that different types of nodes have different feature spaces due to the heterogeneity of nodes. Therefore, a type-specific transformation matrix M_{ψ_i} should be used to project the semantics of logs and entities into the same feature space $x_i = M_{\psi_i} \cdot v_i$, where v_i denotes the representation of the i-th node in the graph, ψ_i denotes the type of node, and x_i denotes the projected feature vector. Then, for each neighbor k on meta-path $\phi \in \Phi$, the attention score $\alpha_{i,j}^{\phi}$ between node i and its neighbor j can be calculated by the following formula:

$$\alpha_{i,j}^{\phi} = \frac{\exp(\sigma(w_{\phi}^{T} \cdot [x_i || x_j]))}{\sum_{k \in \mathcal{N}_i^{\phi}} \exp(\sigma(w_{\phi}^{T} \cdot [x_i || x_k]))}, \tag{3}$$

where $\mathcal{N}_i^{\phi}$ denotes the neighbors of the i-th node on meta-path ϕ, w_{ϕ} denotes the attention vector for ϕ, σ denotes the activation function, and $||$ is the concatenate operation. Then, the meta-path based representation z_i^{ϕ} of the i-th node can be aggregated by the projected features of its neighbor nodes $\mathcal{N}_i^{\phi}$ with attention scores. As suggested in [37], we use multi-head attention mechanism with K heads to make the training process more stable:

$$z_i^{\phi} = \Big\Vert_{k=1}^{K} \sigma\left(\sum_{j \in \mathcal{N}_i^{\phi}} \alpha_{i,j}^{\phi} \cdot x_j \right). \tag{4}$$

Semantic-Level Attention. In a heterogeneous graph, different meta-paths contain different semantic information. Therefore, semantic-level attention mechanism is required for feature aggregation. Considering that logs contain temporal attributes that entities do not have, we first adopt a Gated Recurrent Unit (GRU) model to aggregate the representations $z_i^{\phi}(Log)$ of log nodes in a given meta-path ϕ. For entity nodes, the representations $z_i^{\phi}(Entity)$ are directly aggregated through a Self-Attention layer. Finally, the representations of different meta-paths are aggregated by the semantic-level attention layer, which can be described as:

$$\lambda^{\phi} = \frac{\exp(\tanh(W_s \cdot z^{\phi} + b_s))}{\sum_{\rho \in \Phi} \exp(\tanh(W_s \cdot z^{\rho} + b_s))}, \tag{5}$$

$$Z = \sum_{\phi \in \Phi} \lambda^{\phi} \cdot z^{\phi}, \tag{6}$$

where W_s, b_s denote the weight matrix and the bias vector in semantic-level attention layer, respectively. z^{ϕ}, z^{ρ} denote the matrix composed of node representations z_i^{ϕ} and z_i^{ρ} on corresponding meta-path. Z denotes the aggregated representation of the whole heterogeneous graph. Φ is the complete set of meta-paths.

E. Anomaly Detection

The final representations Z can be further utilized for anomaly detection. Note that Logragh has the capacity to be compatible with different training schemes. In this paper, we treat anomaly detection as a binary classification task and train our model in a supervised manner. Logragh leverages a fully-connected layer and a softmax function to output an anomaly score for each sample:

$$\hat{y} = softmax(W_o \cdot Z + b_o), \tag{7}$$

where W_o, b_o denote the weight matrix and the bias vector, $\hat{y}$ is the anomaly score predicted by the model, indicating the probability that an anomaly occurs. If the anomaly score of a given log sequence is higher than a threshold (e.g., 0.5), then it will be detected as anomalous and an alarm will be raised. Our model is trained based on the Adam (Adaptive Moment Estimation) optimizer, and we leverage cross-entropy as the loss function:

$$\mathcal{L} = -y^T \cdot \log(\hat{y}) - (1-y)^T \cdot \log(1-\hat{y}), \tag{8}$$

where y is the real labels of the inputs.

IV. Experiments

A. Experimental Settings

To evaluate the performance of the proposed model, we conduct a series of experiments to answer the following research questions:

RQ1: How effective is Logragh on grouped log datasets compared with existing anomaly detection models?

RQ2: Does the proposed method have significant advantages in anomaly detection on non-grouped log datasets?

RQ3: How does the result of entity extraction affect the performance of anomaly detection?

RQ4: How much contribution does the Log-Entity Graph make to the improvement of anomaly detection?

RQ5: Is the time consumption of the proposed anomaly detection model acceptable in practice?

RQ6: How interpretable are the anomalies detected by Logragh?

1) Datasets: We conduct our experiments on three most widely-used public datasets Hadoop Distributed File System log (**HDFS**), Blue Gene/L supercomputer log (**BGL**) and Thunderbird supercomputer log (**TDB**) obtained from LogHub [38]. These datasets are also used in existing works such as [9], [12], [18], [21]. The detailed descriptions of the datasets are listed as follows:

TABLE II
Detailed Statistics of Public Log Datasets

Index	HDFS	BGL	TDB
Total # of Log Nodes	11,175,629	4,747,963	10,000,000
Total # of Entity Nodes	870,539	990,859	200,522
Avg. Degree of Logs	5.16	4.86	8.37
Avg. Degree of Entities	40.62	13.73	317.51
# Grouped Samples	575,062	69,155	-
Anomalous Samples %	2.9%	45.4%	-
Avg. # of Logs / Sample	19.8	68.6	-
# Non-grouped Samples	374,119	144,168	300,587
Anomalous Samples %	7.6%	14.8%	24.5%
Avg. # of Logs / Sample	71.5	60.18	72.69

HDFS contains 11,175,629 log messages produced by Amazon's EC2 nodes. The logs can be grouped into 575,062 blocks according to $block\_id$. Each block is manually labeled as normal or anomalous by Hadoop domain experts. There are 16,838 (about 2.9%) blocks related to system anomalies.

BGL contains 4,747,963 log messages collected by Lawrence Livermore National Labs. Each log message is manually labeled as normal or anomalous by domain experts. There are 348,460 (about 7.3%) anomalous logs. The logs can be grouped by $node\_id$, and a log group is labeled as anomalous if it contains at least one anomalous log.

TDB is a large-scale log dataset collected from a Thunderbird supercomputer at Sandia National Labs, which contains more than 200 million log messages. Each log message is manually labeled as normal or anomalous. Grouping strategy is not applicable since this dataset contains multiple grouping fields *user_id*, *component_id*, and there exists heterogeneity in each field. Also, this dataset contains more than 200 million log messages. We use the same 10 million continuous logs to reduce computational overhead as in [18].

To construct **non-grouped datasets**, we slice the timeline into small intervals, and extract the entities that appear in each time interval. Then, we collect all the logs containing these entities, and sort them by timestamp to obtain an interleaved log sequence. A log sequence will be labeled as anomalous if it is related to at least one anomalous log (for BGL and TDB) or anomalous block (for HDFS). The statistics of the datasets used in our experiments are summarized in Table II.

In addition to synthetic datasets, we also test Logragh on real-world industrial dataset, where the interleaving problem is significant. The dataset consists of 27.54 million log messages collected from a large-scale distributed system of an anonymous network service operator. The anomalies in the system are manually identified by IT engineers, which include user operation anomalies, machine running status anomalies, as well as network anomalies. The heterogeneity of the anomalies makes grouping strategies inapplicable and the logs have to be processed in an interleaved form.

2) Baselines: We use the following 9 log-based anomaly detection models as baselines, including statistical models PCA, KNN, unsupervised / semi-supervised sequence models DeepLog, LogAnomaly, PLELog, supervised graphical model GAT, LogGD, and supervised sequence models LogRobust,

NeuralLog. The implementations of these models are publicly available from Github repositories [39], [40], [41], [42].

PCA [43] is a widely-used machine learning algorithm for dimension reduction. Anomalous logs are detected based on the projection of template count vector on normal and abnormal feature spaces.

DeepLog [8] is an unsupervised model which adopts an LSTM model to learn the patterns of normal log sequences. Then it predicts the next event template that will appear. If the actual log template is not included in the top-K predicted candidates, the log sequence will be detected as anomalous.

LogAnomaly [21] is also an LSTM-based unsupervised model that leverages log semantic representations and template count vectors to predicts the next log template. It particularly focuses on synonyms and antonyms when training word vectors. LogAnomaly also uses top-K candidate templates to detect anomalies.

PLELog [22] is a semi-supervised anomaly detection model which leverages Positive-Unlabeled Learning to generate probabilistic pseudo-labels for unlabeled logs. Then it trains an attention-based GRU model to classify the log sequences.

KNN [20] is an efficient machine learning method, which assigns labels for unseen log sequences based on the nearest labeled sequences in the training set.

GAT [44] is a graph-based supervised anomaly detection model. It converts log sequence into an event transition graph, which reflects the relative positions of the event templates in the log sequence. Graph Attention Network is leveraged to learn the graph representations and detect the anomalies.

LogGD [25] uses Graph Transformer Networks to detect anomalies on an event transition graph. It uses cross-entropy as loss function and trains the model in a supervised manner.

LogRobust [9] is a supervised model that regards anomaly detection as a binary classification task. It adopts Fast-Text model to generate word vectors, and trains a bi-directional LSTM model to classify log sequences.

NeuralLog [11] is a supervised anomaly detection model which utilizes Word-Piece tokenization instead of log parsing. It leverages BERT encoder to obtain word vectors and trains a Transformer-based classification model to detect anomalous log sequences.

3) Evaluation Metrics: To measure the effectiveness of log-based anomaly detection, we use the same evaluation metrics as most existing work [9], [11], [21], [22], [25], [28], which are defined as follows:

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}, \tag{9}$$

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}, \tag{10}$$

$$\text{F1-score} = \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}, \tag{11}$$

where **Precision** is the proportion of correctly detected anomalous samples among all detected anomalies, **Recall** is the proportion of correctly detected anomalous samples among all real anomalies, **F1-score** is the harmonic mean of Precision and Recall. TP (True-Positive) denotes the number of correctly detected anomalous log sequences, FP (False-Positive) denotes the number of normal log sequences incorrectly detected as anomalies, FN (False-Negative) denotes the number of undetected anomalies.

4) Parameter Settings: For each dataset, we use 70% samples for training, 10% for validation and 20% for testing. The samples are sorted by timestamp, and the training set consists of samples with earlier time. For the sake of fairness, we make sure that the testing set used for both sequence-based model and graph-based model is exactly the same. That is, the inputs to each model are raw log messages. The only difference is how the logs are processed and utilized. For each model, we refer to the parameter settings given in the original paper, and search for the optimal hyper-parameters based on validation set. We repeat the experiment for five times and use the median F1-score as the result. In our model, the hidden size is set to 128, the node-level attention size is set to 8, and the thresholds for entity extraction are set to 2. We adopt a learning rate of 0.001 and a dropout rate of 0.2 to train our model. The training will be early stopped if there is no loss improvement within 10 consecutive iterations. We implemented our model based on Python 3.7.4 and PyTorch 1.8.1. The experiments are conducted on a server with 32 Intel(R) Xeon(R) 2.60GHz CPUs and Ubuntu 16.04 installed. The source code and datasets are publicly available at Github repository[1].

B. Accuracy on Grouped Datasets (RQ1)

We first evaluate the effectiveness of each model explicit grouping strategy is enabled. The experimental results are listed in the right part of Table III, where the top 4 rows correspond to unsupervised and semi-supervised models, and the bottom 6 rows correspond to supervised models. The highest result in each column is highlighted in bold. Overall, supervised models are superior to unsupervised models in accuracy since they make full use of manual labels. Most of the existing methods can achieve excellent accuracy (i.e., close to 1.0) on grouped datasets. This is because grouping strategy provides access to complete, ordered log records about each entity. For a normal system, there is a clear regularity in the order of events obtained by log grouping strategies. The presence of any type of anomalies will disrupts this regularity and will be easily detected. With the setting where grouping fields can be specified manually, Logragh can achieve the same level of accuracy as state-of-the-art models LogRobust and NeuralLog on grouped datasets.

C. Accuracy on Non-Grouped Datasets (RQ2)

As aforementioned, grouping strategy has brought significant improvements to existing methods. However, if grouping strategies are not applicable, these methods seem to be less effective. The experimental results on non-grouped datasets are shown in the left part of Table III. It can be observed that Logragh is still able to achieve the highest average F1-scores on non-grouped datasets, while the accuracy of other

[1]https://github.com/GeorgeChu2019/Logragh

TABLE III
COMPARISON OF ACCURACY OF DIFFERENT ANOMALY DETECTION MODELS

Category	Model	Non-grouped Dataset (Interleaved Logs)										Grouped Dataset						
		HDFS			BGL			TDB			AVG	HDFS			BGL			AVG
		P	R	F1	P	R	F1	P	R	F1	F1	P	R	F1	P	R	F1	F1
Unsupervised / Semi-supervised	PCA	0.42	0.36	0.39	0.38	0.63	0.47	0.36	0.85	0.50	0.454	0.80	0.65	0.72	0.55	0.84	0.66	0.691
	DeepLog	0.61	0.41	0.49	0.62	0.57	0.59	0.69	0.51	0.59	0.554	0.87	0.97	0.91	0.92	0.86	0.89	0.902
	LogAnomaly	0.55	0.48	0.51	0.59	0.78	0.68	0.43	0.65	0.55	0.580	0.91	0.98	0.94	0.93	0.87	0.90	0.922
	PLELog	0.77	0.67	0.71	0.85	0.78	0.81	0.60	0.94	0.74	0.753	0.90	0.98	0.94	0.85	0.86	0.85	0.896
Supervised Methods	LogRobust	0.76	0.82	0.79	0.86	0.94	0.90	0.91	0.94	0.92	0.871	0.97	0.96	0.97	0.97	0.98	0.97	0.970
	GAT	0.88	0.87	0.87	0.79	0.92	0.85	0.96	0.92	0.94	0.888	0.91	**0.99**	0.95	0.97	**0.99**	**0.98**	0.963
	KNN	0.85	**0.90**	0.88	0.79	0.89	0.83	0.95	**0.97**	0.96	0.890	0.97	**0.99**	**0.98**	0.96	0.98	0.97	0.975
	LogGD	0.94	0.80	0.86	0.88	0.95	0.91	0.91	0.96	0.94	0.905	0.91	**0.99**	0.95	**0.98**	**0.99**	**0.98**	0.966
	NeuralLog	0.88	0.85	0.86	0.89	**0.98**	0.93	0.91	**0.97**	0.94	0.910	0.96	**0.99**	0.97	**0.98**	0.98	**0.98**	**0.976**
	Lograph	**0.95**	**0.90**	**0.93**	**0.96**	0.95	**0.95**	**0.98**	**0.97**	**0.98**	**0.951**	**0.98**	**0.99**	**0.98**	**0.98**	0.96	0.97	0.975

methods has overall declined. More specifically, the average F1-score of LogRobust and NeuralLog drops by 10.3% and 4.1%, respectively. Such decline is mainly caused by the interleaving problems, as introduced in Subsection II-D. Compared with NeuralLog, Lograph improves the F1-score by 8.1% on HDFS, and 2.2% on BGL. For dataset TDB, there are two possible grouping fields *node_id* and *component_id*, neither of which can be used for grouping because the values of the fields are heterogeneous. Besides, it is pointed out in [20] that **data leakage** is the cause of high accuracy of existing methods. However, the interleaving problem disrupts the regularity of log sequence. For example, the data leakage rate (i.e., the proportion of template combinations already seen in training set but appearing again in testing set) is 0.96 on grouped HDFS dataset, but only 0.29 on non-grouped HDFS dataset.

On industrial dataset, Lograph is able to detect 92% of anomalies, with an false alarm rate of 15%. The F1-score achieved on this dataset is 0.88, while existing methods cannot obtain an F1-score higher than 0.76, as shown in Fig. 6. For existing methods, interleaving problem makes the emergence of subsequent event template variable and unpredictable, and thus training schemes such as *Next Prediction* becomes less effective on non-grouped datasets. The same is true for conventional graph-based model, since the interleaving problem introduces errors into graph construction and probability estimation. To summarize, the experimental results above demonstrate that semantic association can be used as an alternative to grouping strategies for anomaly detection of interleaved logs.

D. Performance in Entity Extraction (RQ3)

For entity extraction, we manually annotate the reference entities on 16 public datasets. Each dataset contains 2,000 log messages sampled by [45]. These datasets encompass different systems such as supercomputers, operating systems, distributed systems, and standalone software, with high diversity in data formats. The logs are preprocessed and split by white space delimiters. For each token, we annotate whether it is an entity or not, while the model also gives a prediction accordingly. We regard entity extraction as a binary classification task, and use Precision and Recall to evaluate the performance. Here, Precision is the proportion of correctly extracted entities among all extracted entities, while Recall is the proportion of correctly extracted entities among reference entities.

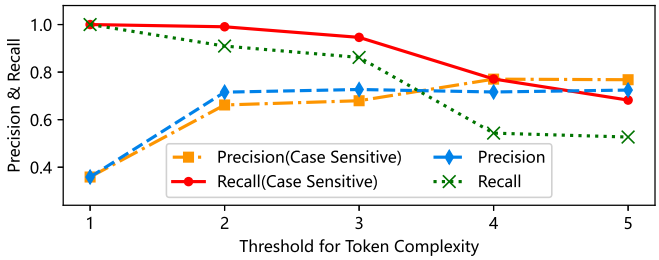


Fig. 5. The impact of the hyperparameters on the average Precision (or Recall) obtained on the 16 datasets with different settings.

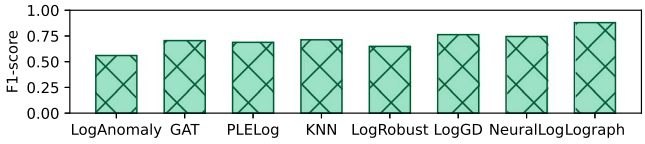


Fig. 6. The F1-score of each method obtained on industry dataset.

TABLE IV
EVALUATION OF ENTITY EXTRACTION ON EACH DATASET

Dataset	Precision	Recall	Dataset	Precision	Recall
Android	0.359	1.0	Mac	0.574	0.859
Apache	0.822	1.0	OpenSSH	0.910	0.999
BGL	0.806	0.963	OpenStack	0.885	1.0
Hadoop	0.539	0.965	Proxifier	0.538	1.0
HDFS	0.860	1.0	Spark	0.486	1.0
HealthApp	0.426	1.0	TDB	0.751	0.992
HPC	0.975	1.0	Windows	0.273	1.0
Linux	0.639	1.0	Zookeeper	0.889	1.0

We explore the impact of the threshold and model design on the performance, as shown in Fig. 5. It can be observed that the Recall gradually falls from 1.0 as the threshold of *TC* increases, while the Precision first rises and then flattens out. In addition, the introduction of a case-sensitive strategy can improve the overall Recall. To trade-off between Precision and Recall, we finally set this threshold to 2 to avoid missing entities as a priority and possibly improve Precision as a second priority. Table IV lists the evaluation result on each dataset, which shows that the proposed method can achieve extremely high Recalls on most of the datasets.

As a cost of high Recall, low Precision has minor impact on the accuracy of downstream task. To elaborate on this, we train

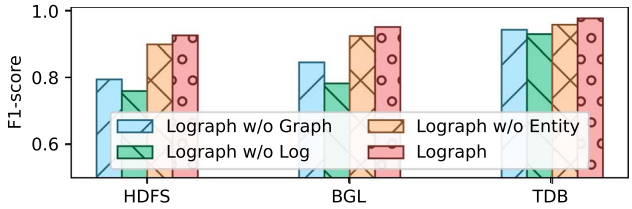


Fig. 7. Results of ablation study on non-grouped datasets. The columns from left to right correspond to Lograph w/o Graph, Lograph w/o Log, Lograph w/o Entity, and Lograph.

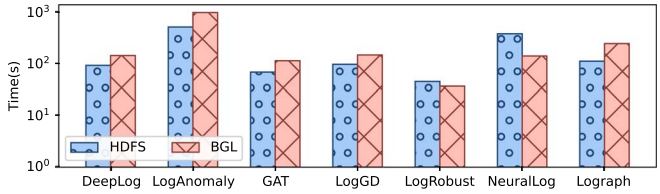


Fig. 8. The average time spent in anomaly detection for every one million log messages.

a binary classifier based on SVM [46] for entity extraction. Although such machine learning based method can achieve an average F1-score 7.7% higher than our approach in entity extraction, its average F1-score in anomaly detection is 2.6% lower than ours. This is because some crucial entities are missed by SVM. For example, node identifiers in the same format as "R63-M1-N0" in dataset BGL are not recognized as entities, since they do not appear in the training set. That is, a high accuracy in entity extraction does not necessarily mean a high accuracy in anomaly detection. Overall, redundant entities mainly affect the efficiency of our model, while missing entities can degrade our model to a sequence model, with a larger impact on downstream tasks. We will further discuss this issue in Section V.

E. Ablation Study (RQ4)

To further evaluate how much contribution the Log-Entity Graph makes to the accuracy of anomaly detection, we conduct ablation tests on the following degraded models:

Lograph w/o Graph does not build a graph structure, but adopts a pure sequence learning model GRU (a sub-module of Lograph) instead. The structure of this degraded model is similar to traditional sequence-based models such as PLELog.

Lograph w/o Entity uses only the representations of log nodes in Log-Entity Graph to detect anomalies. To implement this degraded model, we remove all the components related to entity nodes including *entity-log-entity* meta-path.

Lograph w/o Log uses only the representations of entity nodes to detect anomalies. Similar to the previous degraded model, we remove all the components related to log nodes including *log-entity-log* and *log-log-log* meta-paths.

The experimental results are shown in Fig. 7. It can be observed that removing Log-Entity Graph will cause the accuracy of our model to drop to a level similar to traditional sequence-based methods. The average F1-score obtained by *Lograph w/o Graph* on non-grouped datasets is 0.86, which is 9.5% less than that of Lograph. Furthermore, entity representations provide an average of 2.6% improvement on accuracy, while log representations provide an average of 15.9% improvement. Nevertheless, *Lograph w/o Entity* is also superior to existing methods. For another degraded model *Lograph w/o Log*, there is a significant drop in accuracy due to the lack of crucial information. That is, the representations of the logs themselves play the most important role in anomaly detection. In summary, the superiority of Lograph is attributed to the integration of logs and entities (as well as corresponding meta-paths). The absence of either module will reduce the effectiveness.

F. Evaluation on Efficiency (RQ5)

Efficiency is also an important metric in practice since system anomalies need to be detected and resolved as early as possible. In this subsection, we evaluate the average time spent in anomaly detection for every one million log messages. As illustrated in Fig. 8, the efficiency of anomaly detection is mainly affected by the complexity of the model (e.g., the number of network layers and dimensions). The introduction of Log-Entity Graph can lead to higher accuracy, but also reduce the efficiency as a cost. Nevertheless, the time complexity of our model is still close to that of transformer-based model NeuralLog. To further explore the impact of each module on efficiency, we record detailed time consumption on dataset BGL. Lograph takes an average of 203.18 seconds to detect every one million logs, of which about 47% is spent in node-level attention layers, and about 31% is spent in graph construction and semantic representation. The remaining time is mainly spent in semantic-level attention layer and classification layer. For computational resources, the average **RAM** overhead for storing Log-Entity Graph of HDFS dataset is 148.1MB, and the maximum is 819.7MB, if a 30-minute long sliding window is used. For BGL, the maximum RAM overhead is only 125.5MB and the average is less than 1MB.

In addition, we analyze the speed of log generation for reference. On dataset BGL, the average generation speed is about 450 logs per minute, with a peak of 71,482 logs per minute. On dataset HDFS, the average speed of log generation is about 4,813 logs per minute, with a peak of 210,778 logs per minute. Based on the results in Fig. 8, the **throughput** (i.e., average anomaly detection speed) of Lograph is 295,298 logs per minute and 447,617 logs per minute on these two datasets, respectively, which are even higher than the maximum log generation speeds. When we deploy Lograph for real-time anomaly detection in practice, the following additional optimizations are employed to reduce the computational overhead, though at the cost of a slight (less than 1%) drop in accuracy:

(1) *Stricter entity extraction strategy*. Limit the maximum number of entities to reduce the graph scale, and remove the entities with high correlation to the others.

(2) *Caching mechanism*. Store log representations of each event template, adopt a fixed-interval update strategy for entity representations instead of real-time update.

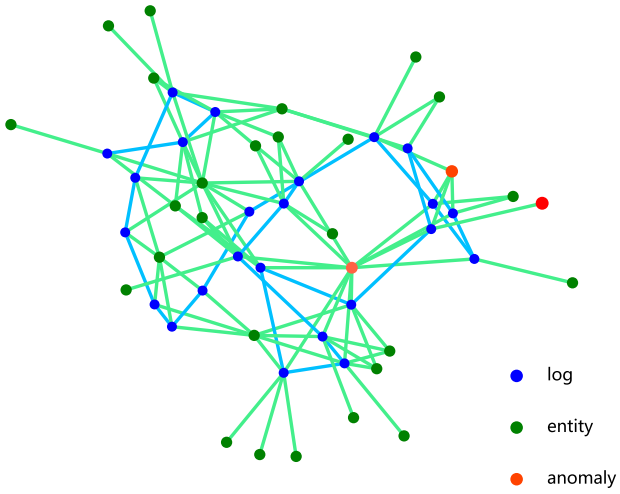


Fig. 9. Visualization of a log-entity graph constructed on an HDFS log fragment, in which the blue points denote log nodes, the green points denote entity nodes, and the red points denote three entities with the highest attention score. The blue lines and the green lines in this figure represent temporal association and semantic association, respectively.

G. Case Study (RQ6)

By mining semantic association, Logragh is able to implicitly group the log messages and detect anomalous in multiple entities. In this subsection, we elaborate on a specific case in HDFS dataset and explore the role of the proposed Log-Entity Graph in anomaly detection. Fig. 9 illustrates the structure of a Log-Entity Graph constructed on an anomalous log sequence, which contains three different blocks, and the related log messages are interleaved together. The anomaly is caused by an IO exception while writing to block $blk\_3163677194922629$. The other blocks in this sequence are in the normal status. To enhance the readability of Fig. 9, we only intercept a continuous sub-sequence at the location of the anomaly, including 24 log nodes, 29 entity nodes and 108 edges.

According to the design of Logragh (Section III-D), the attention score assigned to each entity in Semantic-Level Attention layer indicates the importance of this entity in anomaly detection. A higher attention score implies that the entity is likely to be instrumental for detecting anomalies. That is, the model is confident whether the entity is normal or anomalous. In the example above, the three entities that have the highest attention scores are "$java.io.IOException$", "$blk\_3163677194922629$", and "$dfs.DataNode\$DataXceiver$". The result shows that Logragh successfully identifies the entities most related to the system anomaly. Anomalous entities are located closer to each other on Log-Entity Graph, thus reinforcing each other's abnormal patterns. Also, the attention score of normal blocks are lower than that of the anomalous block. Due to the lack of relevant datasets, we have only a brief exploration here.

Overall, the attention score provides interpretability for anomaly detection and may facilitate fault localization. In future work, the network structure and the loss function of Logragh can be further modified to output anomaly scores for each entity. Logragh can also be utilized for entity anomaly detection task, which can be more practical than sequence-based anomaly detection task.

V. Discussion

In this section, we discuss the following issues:

How missing entities affect our model? Missing entities can reduce the number of edges in the Logragh, which in turn leads to the degradation of the graph-based model into a sequence-based model. Typical entities that are missed by our model are *usernames* composed of purely English letters (e.g., "root", "admin"). Although these entities are difficult to be distinguished from natural language words in appearance, variable parameters in the log parsing results can be utilized for correction. Also, context-based language models may be another solution to this issue.

How redundant entities affect our model? Redundant entities will increase the graph scale, and mainly affect the efficiency of our model. Among all the 16 datasets, *date* and *time* are the most common redundant entities since these two fields are included in every log message. Firstly, *date* (e.g., "2020.11.15") is connected with all the logs in the sequence. It plays a similar role with temporal edges that ensure the connectivity of the whole graph, and add a bias term to the representation of each node. Secondly, *time* (e.g., "21:08:29") is hardly connected with other logs due to its randomness, and has trivial impact on the graph structure. The same is true for low-frequency redundant entities such as hexadecimal random numbers. In addition, compound words (e.g., "Plug-in", "non-protected") and method names (e.g., "nw_nat64_post_new_ifstate", "MTActuatorOpen") can also be incorrectly extracted as entities. However, these tokens are often bound to event templates, and actually play a role in connecting the same templates (which have similar semantic representations). As a result, the representations of these tokens are hardly different from that of the corresponding log nodes. Besides, due to the attention mechanism, our model can ignore edges that are not helpful for downstream tasks.

Is there an alternative entity extraction scheme? There are also alternative approaches such as neural networks. We try to train a context-based BERT model to classify the tokens. Although this model obtains significantly higher F1-scores for entity extraction, it cannot perform well on downstream task due to the absence of one or more "critical" entities. Even if 95% non-critical entities are correctly extracted, but the remaining 5% critical ones are missed, then the related anomalies still cannot be detected. In contrast, a model with a recall of 99% and a precision of 70% may be superior in anomaly detection since it finds more critical entities. Unfortunately, it is difficult to foresee which entities will become critical before the anomaly occurs. After trading off between omission and redundancy, we conclude that a high recall is necessary to avoid model degradation. Therefore, we ultimately adopt the proposed simple but effective entity extraction method for the sake of higher recall and interpretability.

Threats to validity. In terms of external validity, our experiments are conducted on three most commonly used log datasets due to the lack of available public datasets. However,

all the existing datasets are outdated, and anomalies in those datasets are easy to detect. As a result, the accuracy of anomaly detection may decrease when applied to complex and latest datasets. In the future, we will validate the effectiveness of our model on various log datasets. In terms of internal validity, we conduct ablation study to ensure each component of our model contributes to the results. However, current entity extraction and meta-path selection strategies may not be the optimal solution. Meanwhile, real interleaved log data can be more complex than constructed non-grouped data. Future work can focus on the improvement of these components.

VI. Related Work

A. Sequence-Based Anomaly Detection

Supervised models can achieve high accuracy, but require a large number of labeled data. Many deep learning models have been proposed to classify log sequences based on the semantic representation of logs. Specifically, LogRobust [9] leverages a Bi-LSTM model to detect anomalies. Although LogRobust mitigates the effects of unstable logs (including changes in order), it is limited to the case of low injection rates. HitAnomaly [10] adopts a hierarchical Transformer model that focuses on both template words and variable parameters. NeuralLog [11] is also a Transformer-based model, and tries to mitigate the impact of log parsing errors.

Unsupervised models do not rely on anomalous labels, but often fail to accurately understand the essence of anomalies. DeepLog [8] leverages LSTM to predict the subsequent event templates. PCA [43] detects anomalies based on the projection of template count vector in feature space. LogAnomaly [21] is an LSTM-based model which detects both sequential and quantitative anomalies by predicting subsequent event templates. LogSy [47] is a classification-based anomaly detection method, which utilizes self-attention mechanism and auxiliary datasets to improve the log vector representation. LogBert [12] is a self-supervised BERT model trained on masked event template sequence, which also focus on the improvements in representation. RAPID [48] leverages token-level information instead of sequence-level information to distinguish anomalous logs from normal logs. Besides, Auto-Encoder [49] can also be used for log-based anomaly detection. CAT [50] is a self-attentive encoder-decoder transformer framework, which leverages semantic awareness to learn event sequence representations. One-class objective and sequence reconstruction loss are used to train the model.

Semi-supervised models combine the advantages of the above two categories, trading off between accuracy and practicality. LogClass [51] uses PU Learning and Random Forest to detect single log anomaly based on partially labeled data. PLELog [22] adopts Probabilistic Label Estimation strategy to generate pseudo labels for log sequences, and leverages GRU model to detect anomalies.

B. Graph-Based Anomaly Detection

Graphical models can be used not only to discover anomalies in graph topology and node attributes, but to detect anomalous sequences as well [52]. However, these models only focus on the **transition between event templates**, ignoring semantic association. [53] proposes a novel method to detect structural events from log messages to build a directed workflow graph. OASIS [54] converts log template sequences into a control flow graph spanning distributed components. LogFlash [16] utilizes the time-weighted control flow graph and the transition likelihood of each template pair to detect anomalies. The interleaving problem is pointed out in LogFlash, but the main focus of that paper is real-time performance instead. LogGD [25] is also a novel graph-based model which leverages Graph Transformer Networks to detect anomalies. Although these existing works adopt graph-based models, their graph structure is fundamentally different from Lograph.

For other graph structures, DeepTraLog [28] leverages the relationship between traces and logs, and uses gated GNN and deep SVDD for micro-service anomaly detection in the field of AIOps. HiLog [31] integrates human knowledge for enhancement, but it cannot be applied to deep learning models. GraphAD [26] utilizes GNN for entity-wise multivariate time-series anomaly detection.

For **heterogeneous graphs**, HGATE [55] is a graph auto-encoder model and HGAN [37] is a graph attention network, which are applied to general graph anomaly detection tasks, but lack adaptation for specific domains. Related to our work, heterogeneous graph is also used in Log2Vec [56], which designs 10 rules to connect the log entries into a graph and uses thresholds to detect anomalies. The graph used in Log2Vec contributes to mining intrinsic correlations between logs, but the graph structure is different from ours. It does not extract entities, thus the interleaving problem still cannot be solved. Also, the rules are specially customized for typical attacks of insider employees, making it less applicable to datasets other than user operation logs.

VII. Conclusion

In this paper, we propose Lograph, a novel log anomaly detection model based on semantic association mining and heterogeneous graph. To the best of our knowledge, this is the first work that exploits semantic association between logs. The main motivation of this work is to find a feasible solution to anomaly detection of interleaved logs in cases where grouping strategies are not applicable. First, we propose an automated entity extraction approach to obtain the entities. Then, we design a graph structure named Log-Entity Graph to implicitly group the logs. After that, a Heterogeneous Graph Attention Network is trained to detect anomalous logs based on semantic representations of log and entity nodes.

We evaluate Lograph on three public datasets and one real-world industrial dataset, comparing with nine existing anomaly detection models. The results demonstrate that Lograph is able to achieve high accuracy on grouped datasets, and improve the average accuracy by 2%-8% on non-grouped datasets where the logs are interleaved together. The semantic association can be used as an alternative to grouping strategies when the fields are heterogeneous, and is capable of handling the log interleaving problem. We further visualize the graph structure and discuss

some issues about the feasibility and effectiveness of our model. The entity attention score in Lograph provides interpretability for our model and facilitates fault localization. Also, as a data management and feature engineering mechanism, the proposed Log-Entity Graph can be combined with a variety of anomaly detection models, which provides new insights for future works.

REFERENCES

[1] S. He et al., "An empirical study of log analysis at Microsoft," in *Proc. 30th ACM Joint Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng. (ESEC/FSE)*, 2022, pp. 1465–1476.

[2] X. Zhao, Z. Jiang, and J. Ma, "A survey of deep anomaly detection for system logs," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2022, pp. 1–8.

[3] S. Verberne, B. Arends, W. Kraaij, and A. de Vries, "Longitudinal navigation log data on a large web domain," in *Proc. 39th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, New York, NY, USA: ACM, 2016, pp. 697–700.

[4] A. R. Chen, T.-H. Chen, and S. Wang, "Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs," *IEEE Trans. Softw. Eng.*, vol. 48, no. 8, pp. 2905–2919, Aug. 2022.

[5] X. Song, Y. Zhu, J. Wu, B. Liu, and H. Wei, "ADOps: An anomaly detection pipeline in structured logs," *Proc. VLDB Endow.*, vol. 16, no. 12, pp. 4050–4053, Aug. 2023. [Online]. Available: https://doi.org/10.14778/3611540.3611618

[6] A. Vervaet, "MoniLog: An automated log-based anomaly detection system for cloud computing infrastructures," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, 2021, pp. 2739–2743.

[7] S. Elliot, "Devops and the cost of downtime: Fortune 1000 best practice metrics quantified," *Int. Data Cor. (IDC)*, 2014.

[8] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. (CCS)*, 2017, pp. 1285–1298.

[9] X. Zhang et al., "Robust log-based anomaly detection on unstable log data," in *Proc. ESEC/FSE*, 2019, pp. 807–817.

[10] S. Huang et al., "HitAnomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Trans. Netw. Service Manage.*, vol. 17, no. 4, pp. 2064–2076, Dec. 2020.

[11] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2021, pp. 492–504.

[12] H. Guo, S. Yuan, and X. Wu, "LogBERT: Log anomaly detection via BERT," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, 2021, pp. 1–8.

[13] A. Vaswani et al., "Attention is all you need," in *Proc. 31st Int. Conf. Neural Inf. Process. Syst. (NIPS)*, 2017, pp. 6000–6010.

[14] L. Ouyang et al., "Training language models to follow instructions with human feedback," 2022, *arXiv:2203.02155*.

[15] H. Guo et al., "LogFormer: A pre-train and tuning pipeline for log anomaly detection," 2024, *arXiv:2401.04749*.

[16] T. Jia, Y. Wu, C. Hou, and Y. Li, "LogFlash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2021, pp. 80–90.

[17] A. Rebmann, M. Weidlich, and H. Van Der Aa, "GECCO: Constraint-driven abstraction of low-level event logs," in *Proc. IEEE 38th Int. Conf. Data Eng. (ICDE)*, 2022, pp. 150–163.

[18] V.-H. Le and H. Zhang, "Log-based anomaly detection with deep learning: How far are we?" in *Proc. IEEE/ACM 44th Int. Conf. Softw. Eng. (ICSE)*, 2022, pp. 1356–1367.

[19] S. Locke, H. Li, T.-H. P. Chen, W. Shang, and W. Liu, "LogAssist: Assisting log analysis through log summarization," *IEEE Trans. Softw. Eng.*, vol. 48, no. 9, pp. 3227–3241, Sep. 2022.

[20] B. Yu et al., "Deep learning or classical machine learning? An empirical study on log-based anomaly detection," in *Proc. IEEE/ACM 46th Int. Conf. Softw. Eng. (ICSE)*, New York, NY, USA: ACM, 2024.

[21] W. Meng et al., "LogAnomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell. (IJCAI)*, 2019, pp. 4739–4745.

[22] L. Yang et al., "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, 2021, pp. 1448–1460.

[23] Z. Li et al., "Did we miss something important? Studying and exploring variable-aware log abstraction," in *Proc. IEEE/ACM 45th Int. Conf. Softw. Eng. (ICSE)*, 2023, pp. 830–842.

[24] L. Lugo, J. G. Moreno, and G. Hubert, "Segmenting search query logs by learning to detect search task boundaries," in *Proc. 43rd Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, New York, NY, USA: ACM, 2020, pp. 2037–2040.

[25] Y. Xie, H. Zhang, and M. A. Babar, "LogGD: Detecting anomalies from system logs by graph neural networks," 2022, *arXiv:2209.07869*.

[26] X. Chen, Q. Qiu, C. Li, and K. Xie, "GraphAD: A graph neural network for entity-wise multivariate time-series anomaly detection," in *Proc. 45th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, New York, NY, USA: ACM, 2022, pp. 2297–2302.

[27] L. Fang, K. Feng, J. Gui, S. Feng, and A. Hu, "Anonymous edge representation for inductive anomaly detection in dynamic bipartite graph," *Proc. VLDB Endow.*, vol. 16, no. 5, pp. 1154–1167, Jan. 2023. [Online]. Available: https://doi.org/10.14778/3579075.3579088

[28] C. Zhang et al., "DeepTraLog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *Proc. 44th Int. Conf. Softw. Eng. (ICSE)*, 2022, pp. 623–634.

[29] M. Du and F. Li, "Spell: Online streaming parsing of large unstructured system logs," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 11, pp. 2213–2227, Nov. 2019.

[30] G. Chu, J. Wang, Q. Qi, H. Sun, S. Tao, and J. Liao, "Prefix-Graph: A versatile log parsing approach merging prefix tree with probabilistic graph," in *Proc. IEEE 37th Int. Conf. Data Eng. (ICDE)*, 2021, pp. 2411–2422.

[31] T. Jia, Y. Li, Y. Yang, G. Huang, and Z. Wu, "Augmenting log-based anomaly detection models to reduce false anomalies with human feedback," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, 2022, pp. 3081–3089.

[32] B. Dit, L. Guerrouj, D. Poshyvanyk, and G. Antoniol, "Can better identifier splitting techniques help feature location?" in *Proc. IEEE 19th Int. Conf. Program Comprehension*, 2011, pp. 11–20.

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

[34] J. Pennington, R. Socher, and C. Manning, "GloVe: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, Oct. 2014, pp. 1532–1543.

[35] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Oct. 2018, *arXiv:1810.04805*.

[36] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manage.*, vol. 24, no. 5, pp. 513–523, 1988.

[37] X. Wang et al., "Heterogeneous graph attention network," in *Proc. World Wide Web Conf. (WWW)*, 2019, pp. 2022–2032.

[38] "A large collection of system log datasets for AI-powered log analytics." Accessed: May 31, 2022. [Online]. Available: https://github.com/logpai/loghub

[39] "Implementation of PLELog." Accessed: Sep. 19, 2023. [Online]. Available: https://github.com/YangLin-George/PLELog

[40] "Log anomaly detection toolkit." Accessed: Dec. 25, 2023. [Online]. Available: https://github.com/donglee-afar/logdeep

[41] "A toolkit for light automated log anomaly detection." Accessed: Jul. 12, 2024. [Online]. Available: https://github.com/BoxiYu/LightAD

[42] "Implementation of NeuralLog." Accessed: Aug. 21, 2023. [Online]. Available: https://github.com/LogIntelligence/NeuralLog

[43] P. He, J. Zhu, S. He, J. Li, and M. R. Lyu, "Towards automated log parsing for large-scale log data analysis," *IEEE Trans. Dependable Secure Comput.*, vol. 15, no. 6, pp. 931–944, Nov./Dec. 2018.

[44] P. Velikovi, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," 2017, *arXiv:1710.10903*.

[45] S. He, J. Zhu, P. He, and M. R. Lyu, "Loghub: A large collection of system log datasets towards automated log analytics," 2020, *arXiv:2008.06448*.

[46] T. Wu, C. Lin, and R. C. Weng, "Probability estimates for multi-class classification by pairwise coupling," *J. Mach. Learn. Res.*, vol. 5, pp. 975–1005, 2004. Accessed: Sep. 26, 2023. [Online]. Available: http://jmlr.org/papers/volume5/wu04a/wu04a.pdf

[47] S. Nedelkoski, J. Bogatinovski, A. Acker, J. Cardoso, and O. Kao, "Self-attentive classification-based anomaly detection in unstructured logs," in *Proc. IEEE Int. Conf. Data Mining (ICDM)*, 2020, pp. 1196–1201.

[48] G. No, Y. Lee, H. Kang, and P. Kang, "Rapid: Training-free retrieval-based log anomaly detection with PLM considering token-level information," 2023, *arXiv:2311.05160*.
[49] B. Sharma, P. Pokharel, and B. Joshi, "User behavior analytics for anomaly detection using LSTM autoencoder—Insider threat detection," in *Proc. 11th Int. Conf. Adv. Inf. Technol. (IAIT)*, 2020.
[50] S. Zhang, Y. Liu, X. Zhang, W. Cheng, H. Chen, and H. Xiong, "CAT: Beyond efficient transformer for content-aware anomaly detection in event sequences," in *Proc. 28th ACM SIGKDD Conf. Knowl. Discovery Data Mining*, New York, NY, USA: ACM, 2022, pp. 4541–4550. [Online]. Available: https://doi.org/10.1145/3534678.3539155
[51] W. Meng et al., "LogClass: Anomalous log identification and classification with partial labels," *IEEE Trans. Netw. Service Manage.*, vol. 18, no. 2, pp. 1870–1884, Jun. 2021.
[52] Y. Wang, J. Zhang, S. Guo, H. Yin, C. Li, and H. Chen, "Decoupling representation learning and classification for GNN-based anomaly detection," in *Proc. 44th Int. ACM SIGIR Conf. Res. Develop. Inf. Retrieval (SIGIR)*, New York, NY, USA: ACM, 2021, pp. 1239–1248.
[53] F. Wu, P. Anchuri, and Z. Li, "Structural event detection from log messages," in *Proc. 23rd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2017, pp. 1175–1184.
[54] A. Nandi, A. Mandal, S. Atreja, G. B. Dasgupta, and S. Bhattacharya, "Anomaly detection using program control flow graph mining from execution logs," in *Proc. 22nd ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2016, pp. 215–224.
[55] W. Wang et al., "HGATE: Heterogeneous graph attention auto-encoders," *IEEE Trans. Knowl. Data Eng.*, vol. 35, no. 4, pp. 3938–3951, Apr. 2023.
[56] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1777–1794.

Guojun Chu is currently working toward the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications, Beijing, China. His research interests include artificial intelligence for IT operations (AIOps), natural language processing (NLP), anomaly detection, data mining, and pattern recognition.

Jingyu Wang (Senior Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2008. Currently, he is a Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He is a senior member of CIC, and he was selected for the Beijing Young Talents Program. He has published more than 100 papers, such as IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATIONS, IEEE TRANSACTIONS ON MOBILE COMPUTING, IEEE CONFERENCE ON COMPUTER VISION AND PATTERN RECOGNITION, ANNUAL MEETING OF THE ASSOCIATION FOR COMPUTATIONAL LINGUISTICS, ACM INTERNATIONAL CONFERENCE ON MULTIMEDIA, IEEE INTERNATIONAL CONFERENCE ON DATA ENGINEERING, ASSOCIATION FOR THE ADVANCEMENT OF ARTIFICIAL INTELLIGENCE, and so on. His research interests include intelligent networks, edge/cloud computing, machine learning, AIOps and self-driving network, Internet of things (IoV/IoT), knowledge-defined network, and intent-driven networking.

Qi Qi (Senior Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2010. Currently, she is an Associate Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. She has authored or co-authored more than 30 papers in international journals. Her research interests include edge computing, cloud computing, Internet of Things, ubiquitous services, deep learning, and deep reinforcement learning. He was the recipient of two awards from the National Natural Science Foundations of China.

Haifeng Sun (Senior Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. Currently, he is a Lecturer with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. His research interests include artificial intelligence, NLP, big data analysis, object detection, deep learning, and pattern recognition.

Zirui Zhuang (Member, IEEE) received the B.S. and Ph.D. degrees from Beijing University of Posts and Telecommunications, in 2015 and 2020, respectively. Currently, he is a Postdoctoral Researcher with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. In 2019, he visited the Department of Electrical and Computer Engineering, University of Houston. His research interests include network routing and management for next-generation network infrastructures, using machine learning and artificial intelligence techniques, including deep learning, reinforcement learning, graph representation, multiagent systems, and Lyapunov-based optimization.

Bo He (Member, IEEE) received the Ph.D. degree from Beijing University of Posts and Telecommunications, China, in 2023. Currently, he is a Postdoctoral Researcher with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. From 2021 to 2022, he was a Visiting Ph.D. Student with the University of Waterloo, Canada. His research interests include 5G/6G networks, multipath networks, collective communication, transmission control, and deep reinforcement learning.

Yuhan Jing received the master's degree from Beijing University of Posts and Telecommunications, China, in 2020. She is currently working toward the Ph.D. degree with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. Her research interests include AIOps, time series analysis, anomaly detection, and fault localization.

Lei Zhang received the master's degree from Nanjing University of Posts and Telecommunications in 2004. Currently, she is a Technical Expert with the Department of Cloud Network Center, China Unicom, and the Leader of the Digital twin Project. Her research interests include mobile network, network management, AIOps, digital twin, etc.

Jianxin Liao (Senior Member, IEEE) received the Ph.D. degree from the University of Electronics Science and Technology of China, Chengdu, China, in 1996. Currently, he is the Dean of the Network Intelligence Research Center and a Full Professor with the State Key Laboratory of Networking and Switching Technology, Beijing University of Posts and Telecommunications. He has authored or co-authored hundreds of research papers and several books. He has won a number of prizes in China for his research achievements. His research interests include cloud computing, mobile intelligent networks, service network intelligence, networking architectures and protocols, and multimedia communication.