GLAD: Content-aware Dynamic Graphs For Log Anomaly Detection

Yufei Li[1] Yanchi Liu[2] Haoyu Wang[2] Zhengzhang Chen[2] Wei Cheng[2] Yuncong Chen[2] Wenchao Yu[2]
Haifeng Chen[2] Cong Liu[1]
[1]University of California, Riverside [2]NEC Labs America
[1]{yli927,congl}@ucr.edu [2]{yanchi,haoyu,zchen,weicheng,yuncong,wenchao,haifeng}@nec-labs.com

2023 IEEE International Conference on Knowledge Graph (ICKG) | 979-8-3503-0709-2/23/$31.00 ©2023 IEEE | DOI: 10.1109/ICKG59574.2023.00007

***Abstract*—Logs play a crucial role in system monitoring and debugging by recording valuable system information, including events and states. Although various methods have been proposed to detect anomalies in log sequences, they often overlook the significance of considering relations among system components, such as services and users, which can be identified from log contents. Understanding these relations is vital for detecting anomalies and their underlying causes. To address this issue, we introduce GLAD, a Graph-based Log Anomaly Detection framework designed to detect relational anomalies in system logs. GLAD incorporates log semantics, relational patterns, and sequential patterns into a unified framework for anomaly detection. Specifically, GLAD first introduces a field extraction module that utilizes prompt-based few-shot learning to identify essential fields from log contents. Then GLAD constructs dynamic log graphs for sliding windows by interconnecting extracted fields and log events parsed from the log parser. These graphs represent events and fields as nodes and their relations as edges. Subsequently, GLAD utilizes a temporal-attentive graph edge anomaly detection model for identifying anomalous relations in these dynamic log graphs. This model employs a Graph Neural Network (GNN)-based encoder enhanced with transformers to capture content, structural and temporal features. We evaluate our proposed method[1] on three datasets, and the results demonstrate the effectiveness of GLAD in detecting anomalies indicated by varying relational patterns.**

***Index Terms*—log anomaly detection, GNN, transformer**

I. Introduction

Anomaly detection is the task of identifying unusual or unexpected behaviors in a system or process. As computer systems become increasingly more sophisticated due to the expansion of new communication technologies and services, they are prone to various adversarial attacks and bugs [1]. Moreover, such attacks are also getting evolved and becoming increasingly sophisticated. As a result, the difficulty of anomaly detection has increased, making many conventional detection approaches no longer effective, and it requires us to look deeper into the system, for example, the interaction among system components.

System logs capture system states and events across time to aid process monitoring and root cause analysis of running services. These log files are ubiquitous in almost all computer systems and contain rich information, including control commands of machine systems, transactions of customer purchases, and logs of a computer program. As a result, they have proven a valuable resource for anomaly detection in both academic research and industry applications [2]–[6]. Each log message usually consists of a predefined constant key template (known as a "event", e.g., a login activity) and a few variables (known as "fields", e.g, *services* and *users*). When the events are arranged chronologically based on the recording time, they form a discrete log sequence. Various methods have been proposed to detect the anomalous sequential patterns in the sequence: (1) *Pattern recognition* methods consider event sequences with inconsistencies beyond a certain threshold to be anomalous [7]–[11]. They treat event alphabet sequence as input in an independent dimension and ignore the sequential patterns between events. (2) *Sequential learning* methods analyze events sequentially with a defined sliding window in order to forecast the subsequent event based on the observation window [6], [12].

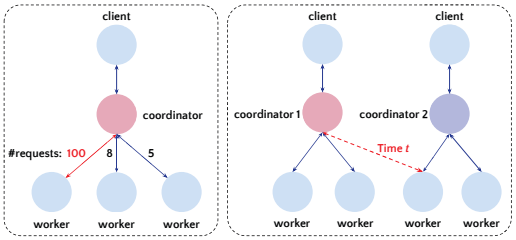


Fig. 1: Two anomalous relations (anomalous edges highlighted in red): unbalanced request (left) and malicious request (right).

However, the relation between log events and fields, an essential indicator of system anomalies, has often been overlooked. This oversight can lead to missed detection or false alarms, as anomalies may not be apparent from individual events or isolated patterns. Different from previous methods that detect anomalous sequential patterns in log sequences, we focus on a new task that aims at detecting anomalous relational patterns between interconnected events and fields. Take, for instance, a scenario where workers receive an unbalanced number of requests from a coordinator in a period of time, or a coordinator suddenly requests connection to other workers, as illustrated in Figure 1. Traditional methods, without considering the relations, may fall short in detecting such anomalies. Apart from detecting anomalous events, understanding these anomalous relations between events can offer insightful details about the system's dynamics, for example, the underlying causes of an anomaly and its propagation over time.

[1]Our code is available at https://github.com/yul091/GraphLogAD

979-8-3503-0709-2/23/$31.00 ©2023 IEEE
DOI 10.1109/ICKG59574.2023.00007

To achieve our goal, there are several challenges: (1) Dynamic graphs need to be built to describe the interactions between log events and fields in different time windows. (2) Instead of merely detecting anomalies on graph level, we aim to detect anomalous edges representing the relations among nodes, which is a more challenging task. (3) In addition to relational patterns, we need to integrate log semantics and sequential patterns as a whole for anomaly detection.

To this end, we propose GLAD, a **G**raph-based **L**og **A**nomaly **D**etection framework, to extract and learn the relations among log events and fields, in addition to log semantics and sequential patterns, for system relation anomaly detection. Our approach proposes a novel method to construct dynamic graphs that describe the relations among log events and fields over time and then leverages a temporal-attentive transformer to capture the sequential patterns implicitly expressed in each time period. Specifically, a field extraction module utilizing prompt-based few-shot learning is first used to extract field information from log contents. Then, with the fields extracted and the log events parsed from a log parser, dynamic graphs can be constructed for sliding windows with events and fields as nodes and the relations between them as edges. Finally, a temporal-attentive graph edge anomaly detection method is proposed to detect anomalous relations from evolving graphs, where a Graph Neural Network (GNN)-based encoder facilitated with transformers is used to learn the structural, content, and sequential features. Experiments on real-world log datasets are conducted to demonstrate the effectiveness of GLAD.

To summarize, in this work, we propose to detect log anomalies from a novel point of view, i.e., the interaction and relation between system components leveraging system logs. In this way, we can dig into more system details and find causes and solutions to the anomalies efficiently. Our main contribution is a framework for constructing dynamic graphs from logs and capturing relational anomalies from dynamic graphs using temporal-attentive transformers, which allows for more granular and accurate log anomaly detection. We believe our proposed approach has the potential to significantly improve the effectiveness of log analysis in detecting more sophisticated anomalies in real applications.

II. Related Work

Log Sequences Anomaly Detection. Detecting anomalies in log sequences has recently gained substantial attention. Earlier research hinged upon similarity measurements, wherein test logs are compared with training logs to detect anomalies based on their dissimilarity [13], [14]. Subsequent methods can be categorized into three groups: *pattern frequency-based* [15], *sequence-based* such as Hidden Markov Model (HMM) [16], and *contiguous subsequence-based* anomaly detection such as window-based techniques [3], [17]. While certain studies utilize supervised learning for anomaly detection [18]–[21], unsupervised learning, which observe only normal event sequences during training, has been proven to be a more efficient learning paradigm [8]–[11], [22], [23]. Our research mainly focuses on the latter learning paradigm.

TABLE I: Notation Description.

Symbol	Description
e	$e = \{x_1, ..., x_{\|e\|}\}$ log message is a sequence of tokens
S	$S = \{e_1, ..., e_{\|S\|}\}$ log sequence is a sequential series of logs
E	$E = \{ent_1, ..., ent_{\|E\|}\}$ sequence of entities in a log message
Y	$Y = \{l_1, ..., l_{\|E\|}\}$ sequence of entity labels in a log message
$\mathcal{S}$	$\mathcal{S} = \{S_1, ..., S_{\|\mathcal{S}\|}\}$ total sequences are a set of log sequences
$\mathcal{G}_t$	the dynamic graph at time window t with $\mathcal{V}_t$ and $\mathcal{E}_t$
$\mathcal{V}_t$	vertex set in graph $\mathcal{G}_t$
$\mathcal{E}_t$	edge set in graph $\mathcal{G}_t$
$\mathbf{X}_t$	attribute matrix in graph $\mathcal{G}_t$
$\mathbf{A}_t$	adjacency matrix in graph $\mathcal{G}_t$
$\mathbf{W}^{(l)}$	learnable weights in the l-th layer of a model, e.g., $\mathbf{W}_{ner}$, $\mathbf{W}_g^{(l)}$
$\mathbf{I}$	identity matrix
$\mathbf{H}_t$	node representations of graph $\mathcal{G}_t$ learned by GCN
N	total number of graphs in $\mathcal{S}$
$\boldsymbol{\mathcal{H}}_{\mathcal{S},t}$	long-term node representations of graph $\mathcal{G}_t$ learned by transformers
$\boldsymbol{\mathcal{H}}_{k,t}$	short-term node representations of graph $\mathcal{G}_t$ learned by transformers
$\boldsymbol{\mathcal{H}}_t$	node representations of graph $\mathcal{G}_t$ by concatenating $\boldsymbol{\mathcal{H}}_{\mathcal{S},t}$ and $\boldsymbol{\mathcal{H}}_{k,t}$
$\boldsymbol{\mathcal{R}}_t$	graph representation of $\mathcal{G}_t$ by maxpooling node representations $\boldsymbol{\mathcal{H}}_t$
$\sigma(\cdot)$	activation function, e.g., ReLU(·), Sigmoid(·)
$\mathcal{L}$	loss objective, including $\mathcal{L}_{ner}$, $\mathcal{L}^t$, $\mathcal{L}_{reg}$
$\mathbf{P}$	prompt $\mathbf{P} = \{p_1, ..., p_m\}$, including $\mathbf{P}^+$ and $\mathbf{P}^-$

Log Knowledge Graph Construction. Raw log files offer a wealth of information pertaining to system states and service interconnection, e.g., whether a computing machine is running under an abnormal state or a user is a malicious attacker. To analyze such data and avoid tedious searching clues or tracing system events across log sources, existing studies have put efforts into identifying and linking entities (log fields) across log sources, thereby enriching them with knowledge graphs [24]–[26]. They often apply information extraction techniques such as Named Entity Recognition (NER) to identify log fields within log messages. The resulting fields are considered nodes within a knowledge graph, and rule-based relation linking is used to integrate the log fields into the knowledge graph. However, these methods require a large amount of label data for training, which introduce high cost in real applications. In comparison, we try to solve this problem in a few-shot setting.

Graph-based Anomaly Detection. GNNs have become increasingly popular due to their ability to learn relation patterns, making them favorable for anomaly detection. Leading GNN models include GCN [27], GIN [28], SAGE [29], GAT [30], and Transformer Graph (GT) [31]. Existing graph-based anomaly detection methods can be categorized into three types based on the range of anomaly detection: (1) *Node-level auto-encoders* [32]–[35] regard nodes with atypical attribute and relation distributions as anomalies. The key idea is to use GNN-based encoder-decoders to reconstruct original graphs and calculate the reconstruction errors for each node. Nodes with above-threshold errors are detected as anomalies. Some further consider temporal relations on dynamic graphs [36], [37] to detect anomalies. (2) *Edge-level auto-encoders* [38], [39] first use graph encoders to learn node feature representations, then determine edge scores for each node pair in the graph to represent how likely it is normal. Some further consider representative structural information from the dynamic graph in each time stamp and their dependencies [36], [40], [41] to detect anomalous edges. (3) *Graph-level auto-encoders* [42]–[46] use a graph encoder to learn feature representations and aggregates all node features within each

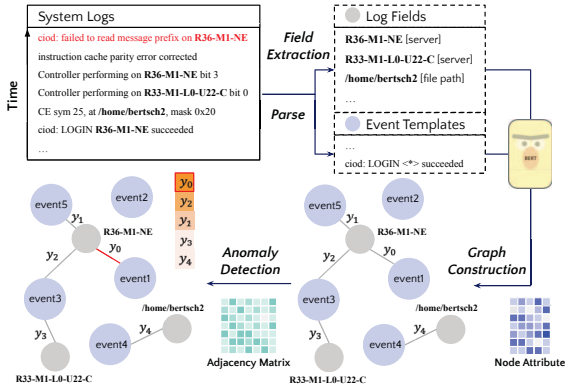


Fig. 2: Overview of our GLAD framework. GLAD first extracts log fields and events and connects them to construct dynamic log graphs, where node features are text embeddings. These graphs, along with their sequential dependencies, are jointly encoded to identify anomalous edges.

graph as the graph representation. Hypersphere learning is then applied to cluster all normal graphs into a central distribution, distinguishing them from anomalous ones.

III. Log Anomaly Detection Framework

In this section, we introduce GLAD, a graph-based framework that learns structural, content, and sequential features among logs for anomaly detection, as shown in Figure 2.

A. Preliminaries

We first define several important terminologies pertinent to our work. The notions are summarized in Table I.

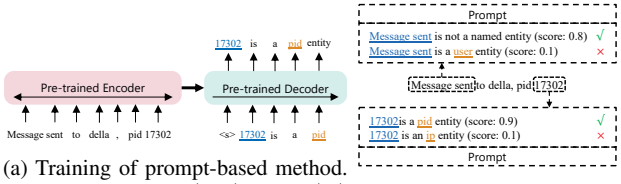
A **log** is a sequence of tokens $e = \{x_1, ..., x_{|e|}\}$, where x_i denotes the i-th token and $|e|$ is the log length.

A **log sequence** is a series of logs ordered chronologically within an observed time window $S = \{e_1, ..., e_{|S|}\}$, where e_i represents the i-th log and $|S|$ denotes the total number of logs in a time window.

For a log sequence S_t in time window t, we construct a **dynamic graph** $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathbf{X}_t, \mathbf{A}_t)$, where $\mathcal{V}_t$, $\mathcal{E}_t$ denote the union of vertices and the union of edges, $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ and $\mathbf{A}_t \in \mathbb{R}^{n \times n}$ are its attribute and adjacency matrices. Note that the dynamic graph used in this paper is an undirected, weighted, and attributed heterogeneous graph.

B. Log Graph Construction

To build graph representations from log sequences, we propose a prompt-based model to extract fields from log messages. The extracted fields, along with the parsed log events via a log parser, are interconnected following predefined principles to construct dynamic graphs. Subsequently, we employ a pre-trained Sentence-BERT [47] to capture the semantics of each node using its content information. The encoded hidden representations for each node are treated as its attributes, while the adjacency matrix represents the structure of the graph. These node attributes and adjacency matrices are collectively used to detect anomalous edges.



(a) Training of prompt-based method. The prompt used is "$\langle x_{i:j} \rangle$ is a/an $\langle l_k \rangle$ entity".

(b) Inference of the prompt-based method.

Fig. 3: Illustration of prompt-based few-shot field extraction.

Prompt-Based Few-Shot Field Extraction. Real-world log datasets contain substantial log events and log fields with diverse syntactic formats, making manual annotations virtually infeasible. While existing off-the-shelf tools [48], [49] employ either rule-based or search-based algorithms to extract event templates and fields from raw log messages, their effectiveness is limited. They work well with fields exhibiting fixed syntax patterns, such as *IP*, *email*, and *URL*, but falter with those that have flexible syntax patterns, like *user* and *service*.

To overcome this challenge, we approach log field extraction as a NER task and propose a prompt-based few-shot learning method using BART [50] that excels in identifying log fields in low-resource scenarios. We define 15 common log field types vital for system monitoring by referring to common log ontology [24]–[26]. These include *IP*, *email*, *process ID* (*pid*), *user ID* (*uid*), *user name*, *timestamp*, *service*, *server*, *file path*, *URL*, *port*, *session*, *duration*, *domain*, and *version*.

We frame the field extraction as a seq2seq learning process, as shown in Figure 3a. Given a log message $e = \{x_1, ..., x_{|e|}\}$, which contains a set of gold fields $E = \{ent_1, ..., ent_{|E|}\}$ and a label set $Y = \{l_1, ..., l_{|E|}\}$, we create a target sequence (prompt) $\mathbf{P}_{l_k, x_{i:j}} = \{p_1, ..., p_m\}$ for each candidate text span $x_{i:j}$ and its label l_k. Specifically, $\mathbf{P}$ is a positive prompt $\mathbf{P}^+$ if the text span is a gold field ($x_{i:j} \in E$), e.g., "$\langle x_{i:j} \rangle$ is a/an $\langle l_k \rangle$ entity"; otherwise, it is a negative prompt $\mathbf{P}^-$, e.g., "$\langle x_{i:j} \rangle$ is not a named entity".

During training, we create prompts using gold fields following [51], [52]. For each log message e, we create positive pairs $(e, \mathbf{P}^+)$ by traversing all its gold fields and negative pairs $(e, \mathbf{P}^-)$ by randomly sampling non-entity text spans. For efficiency, we limit the number of n-grams for a span to 1∼5, i.e., $5{*}n$ negative prompts are created for each log message. After sampling, the number of negative pairs is three times that of positive pairs. Given a sequence pair $(e, \mathbf{P})$, we feed the log message e to the encoder of BART whose hidden size is d_h, and obtain the hidden states $\mathbf{h}^{enc} \in \mathbb{R}^{d_h}$:

$$\mathbf{h}^{enc} = \text{Encoder}(x_{1:|e|}) \tag{1}$$

At the c-th decoding step, $\mathbf{h}^{enc}$ and previous output tokens $p_{1:c-1}$ are used to generate a representation via attention [53]:

$$\mathbf{h}_c^{dec} = \text{Decoder}(\mathbf{h}^{enc}, p_{1:c-1}) \tag{2}$$

The conditional probability of a word p_c is defined as:

$$P(p_c | p_{1:c-1}, e) = \text{softmax}(\mathbf{h}_c^{dec} \mathbf{W}_{ner} + \mathbf{b}_{ner}) \tag{3}$$

where $\mathbf{W}_{ner} \in \mathbb{R}^{d_h \times |V|}$ and $\mathbf{b}_{ner} \in \mathbb{R}^{|V|}$. Here $|V|$ denotes the vocab size of BART. The decoding objective is the Cross-Entropy (CE) loss for prompt with length m:

$$\mathcal{L}_{ner} = -\sum_{c=1}^{m} \log P(p_c|p_{1,c-1}, e) \tag{4}$$

During inference, we enumerate all possible 1∼5-grams text spans $x_{i:j}$ for a log message e and compute scores for each prompt $\mathbf{P}_{l_k,x_{i:j}} = \{p_1, ..., p_m\}$ as follows:

$$f(\mathbf{P}_{l_k,x_{i:j}}) = \sum_{c=1}^{m} \log P(p_c|p_{1:c-1}, e) \tag{5}$$

For each traversed text span $x_{i:j}$, we compute the score $f(\mathbf{P}^{+}_{l_k,x_{i:j}})$ for every entity type and $f(\mathbf{P}^{-}_{x_{i:j}})$ for the non-entity type. A resulting type l^*_k than garners the highest score is assigned to $x_{i:j}$. Such iterative process ensures the extraction of all relevant fields, as depicted in Figure 3b.

Graph Structure Configuration. To model the relation between fields and events across different log messages, we use a sliding window with a fixed time interval to snapshot a batch of log messages and construct a corresponding graph. Specifically, each log instance consists of a parsed event template (obtained via a log parser such as Drain [54]), e.g., "FAILED LOGIN for ⟨∗⟩ to ⟨∗⟩", along with a list of extracted fields, e.g., ["della", "imap://localhost/"] with corresponding types, e.g., [*user*, *server*]. We then interconnect the event template to each extracted field to capture inherent behaviors in the log with the number of connections as edge weight. In the resultant undirected graph, any two log instances that share any of the defined nodes are indirectly connected, thereby indicating their implicit relations.

Graph Node Attribute Configuration. We define types of nodes based on the corresponding event and field types, such as *server* for "imap://localhost/". For each node, we define its input text format and employ a pre-trained Sentence-BERT [47] to learn the sentence embedding as its attribute. Specifically, for log events, we directly use their templates as the encoder input texts, while for log fields we use our defined prompts as the input texts, e.g., "imap://localhost/ is a server entity". The output hidden states for each input text capture the node semantics and are used as node features for constructing attributed graphs.

C. Temporal-Attentive Graph Edge Anomaly Detection

We now introduce our proposed temporal-attentive graph edge anomaly detection method, as illustrated in Figure 4, which operates on the dynamic graphs constructed for logs within corresponding time slots. Specifically, a Graph Convolutional Network (GCN) is first used to encode the structural information for each graph. Then, a transformer encoder is deployed to learn the temporal dependencies within the sequence of dynamic graphs. For each graph, we sample certain negative edges and compute the edge score using the learned hidden states. The process concludes by employing a pair-wise margin loss to minimize the positive edge scores

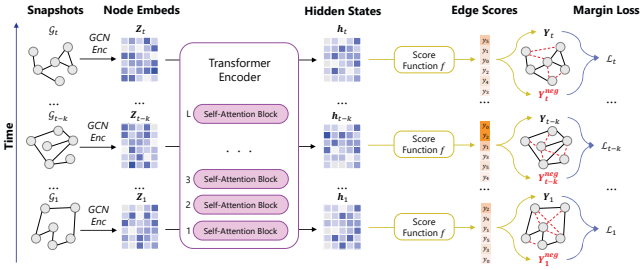


Fig. 4: Overview of our temporal-attentive graph edge anomaly detection framework. Edges highlighted in red are negative edges. $\mathbf{Y}_t$ and $\mathbf{Y}^{neg}_t$ denote the aggregation of positive and negative edge scores, respectively, for a specific graph $\mathcal{G}_t$.

and maximize the negative edge scores, in adherence with the one-class training objective.

GCN Shared Encoder. At time window t, we receive a graph snapshot $\mathcal{G}_t = (\mathcal{V}_t, \mathcal{E}_t, \mathbf{X}_t, \mathbf{A}_t)$, where $\mathbf{X}_t \in \mathbb{R}^{n \times d}$ and $\mathbf{A}_t \in \mathbb{R}^{n \times n}$ represent its attribute and adjacency matrices respectively. We apply GCN [27] to capture both its attribute and structural features. While there exist advanced GNNs, such as Graph Transformer (GT) [31], we found that GCN offers a blend of efficiency and competitive performance. It considers high-order node proximity when encoding the embedding representations, thereby alleviating network sparsity beyond the observed links among nodes [32]. For an L-layered GCN, each layer can be expressed with the function:

$$\begin{aligned} \mathbf{H}^{(l)}_t &= f_\sigma(\mathbf{H}^{(l-1)}_t, \hat{\mathbf{A}}_t|\mathbf{W}^{(l)}_g) \\ f_\sigma(\mathbf{H}^{(l)}_t, \hat{\mathbf{A}}_t|\mathbf{W}^{(l)}_g) &= \sigma(\hat{\mathbf{D}}_t^{-\frac{1}{2}}\hat{\mathbf{A}}_t\hat{\mathbf{D}}_t^{-\frac{1}{2}}\mathbf{H}^{(l)}_t\mathbf{W}^{(l)}_g) \end{aligned} \tag{6}$$

where $\mathbf{W}^{(l)}_g$ is a learnable weight matrix for the l-th layer, $l \in [1, L]$. $\hat{\mathbf{A}}_t = \mathbf{A}_t + \mathbf{I}$ denotes the adjacency matrix with added self-loops and $\hat{\mathbf{D}}_{i,i} = \sum_{j=0} \hat{A}_{i,j}$ represents its diagonal degree matrix. $\sigma(\cdot)$ is a non-linear activation function, for which we use the ReLU.

We designate the attribute matrix $\mathbf{X}_t$ as the initial hidden state $\mathbf{H}^{(0)}_t$. The resultant embedding $\mathbf{Z}_t = \mathbf{H}^{(L)}_t$ captures the nonlinearity of complex interactions between log entities and events within each graph. However, it is still inadequate for detecting anomalies caused by malicious relations due to neglect of temporal features across graph snapshots.

Temporal-Attentive Transformer. Given the chronologically generated nature of system logs, and the logical dependencies that exist between past and present log states, we employ a transformer encoder to incorporate the temporal features of entire sequence into the latent space.

We receive a sequence of node embeddings $\{\mathbf{Z}_1, ..., \mathbf{Z}_N\}$ for all graphs. Note that nodes in each graph are an unordered set, $\mathcal{V}_t = \{v_1, ..., v_{|\mathcal{V}_t|}\}$, rather than a sequence. We propose a Set Transformer (ST) to eliminate this order dependencies when encoding node embeddings. Specifically, we first compute the position embeddings based on each graph's position in the sequence, assigning all nodes belonging to each graph the identical position embedding $\mathbf{E}_p$. Subsequently, the embedding

for the graph at time t (with position p) is determined as $\mathbf{E}_t = \mathbf{E}_p + \mathbf{Z}_t$, and the representation sequence as $\mathbf{E}_{\mathcal{S}} = \{\mathbf{E}_1, ..., \mathbf{E}_N\}$. The representation sequence is then fed into self-attention blocks to derive long-term representations $\boldsymbol{\mathcal{H}}_{\mathcal{S}}$:

$$\boldsymbol{\mathcal{H}}_{\mathcal{S}}^{(l+1)} = \text{FFN}(\text{Attention}(\boldsymbol{\mathcal{H}}_{\mathcal{S}}^{(l)})) \tag{7}$$

where l denotes the layer index, with the initial hidden state $\boldsymbol{\mathcal{H}}_{\mathcal{S}}^{(0)} = \mathbf{E}_{\mathcal{S}}$. We formulate subsequences using a sliding window of size k. Consequently, each subsequence comprises unique local information, pivotal in determining whether the entire sequence is anomalous. For a subsequence of graph node embeddings $\{\mathbf{Z}_{t-k-1}, ..., \mathbf{Z}_t\}$, corresponding to graphs $\{\mathcal{G}_{t-k-1}, ..., \mathcal{G}_t\}$, its representation can be expressed as $\mathbf{E}_k = \{\mathbf{E}_{t-k-1}, ..., \mathbf{E}_t\}$. The same operations are executed to obtain short-term representations $\boldsymbol{\mathcal{H}}_k$ by considering k local graphs.

We then concatenate the encoded long-term $\boldsymbol{\mathcal{H}}_{\mathcal{S}}$ and short-term representations $\boldsymbol{\mathcal{H}}_k$ to form the final node features:

$$\boldsymbol{\mathcal{H}} = [\boldsymbol{\mathcal{H}}_{\mathcal{S}} || \boldsymbol{\mathcal{H}}_k]_{dim=1} \tag{8}$$

where $[\cdot||\cdot]_{dim=1}$ represents the concatenation operator of two matrices over the column-wise dimension. Consequently, the final node representations $\boldsymbol{\mathcal{H}}_t$ for graph $\mathcal{G}_t$ captures the structural, content, and temporal features.

Edge-level Training objective. Until now, we have established the hidden states of nodes $\boldsymbol{\mathcal{H}}_t$ at time window t. For each edge $(i, j, w) \in \mathcal{E}^t$ with weight w, we retrieve the embeddings for the i-th and j-th node in $\boldsymbol{\mathcal{H}}_t$. This allows us to calculate its anomalous score as follows:

$$f(i, j, w) = w \cdot \sigma(\mathbf{W}_1 \mathbf{h}_i + \mathbf{W}_2 \mathbf{h}_j - \mu) \tag{9}$$

where $\mathbf{h}_i$ and $\mathbf{h}_j$ are the hidden states of the i-th and j-th node respectively, and $\sigma(\cdot)$ is the sigmoid function. $\mathbf{W}_1$ and $\mathbf{W}_2$ are the weights in two fully-connected layers. μ is a hyperparameter in the score function. Note that this single layer network can be replaced by more complex networks.

To overcome the scarcity of anomaly data during training, we build a model to optimize one-class (normal) data instead. In essence, this means that all edges are considered normal during training. Inspired by the sampling method proposed in [55], we apply a Bernoulli distribution with parameter $\frac{d_i}{d_i+d_j}$ for sampling anomalous edges according to the node degree d. In particular, for each normal edge (i, j) in the graph, we generate an anomalous edge by either replacing node i with node i' (with a probability of $\frac{d_i}{d_i+d_j}$) or replacing node j with node j' (with a probability of $\frac{d_j}{d_i+d_j}$). Here, d_i and d_j are the degrees of the i-th and j-th node respectively. Realizing that the generated edges may still be normal [41], [56], we propose a margin-based pairwise edge loss in training rather than a strict objective function such as cross entropy, to distinguish between existing edges and generated edges:

$$\mathcal{L}_e = \sum_{t=1,...,N} \min \sum_{(i,j,w)\in\mathcal{E}^t} \sum_{(i',j',w)\notin\mathcal{E}^t} \max\{0, \gamma + f(i, j, w) - f(i', j', w)\} \tag{10}$$

where $\gamma \in (0, 1)$ is the margin between the likelihood of normal and anomalous edges, and $f(\cdot, \cdot, \cdot)$ is the aforementioned anomalous edge score function. Minimizing the loss function $\mathcal{L}_e$ results in a smaller $f(i, j, w)$ and a larger $f(i', j', w)$, thereby achieving our one-class optimization goal.

To enhance efficiency, we aim to select edges of high significance for training. Specifically, for each pair of normal edge (i, j, w) and negatively sampled edge (i', j', w), we discard it if $f(i, j, w) > f(i', j', w)$ and retain it otherwise, for pair-wise optimization. The intuition behind is that some edges in snapshots may not be entirely normal after training, and we aim to increase the reliability of normal edges that are used to learn graph representations. This selective negative sampling paradigm bolsters the stability of GLAD in training.

Multi-granularity Learning. Besides the margin loss that differentiates normal and anomalous edges, we introduce an ad-hoc heuristic to form a "soft-margin" decision boundary. This means we select graph representations whose distance to a center ranks at specific percentile as the decision boundary's radius [12]. To this end, we first formulate the graph representation for $\mathcal{G}_t$ by maxpooling its node representations:

$$\boldsymbol{\mathcal{R}}_t = \text{maxpooling}(\boldsymbol{\mathcal{H}}_t) \tag{11}$$

At the graph-level, anomalous graphs can be detected via one-class classification training. The objective $\mathcal{L}_g$ is to learn a minimized hypersphere that encloses graph representations:

$$\begin{aligned} &\min_{R,\mathbf{c},\varepsilon} R^2 + C\sum_{t=1}^{N} \varepsilon_t \\ &s.t.\ ||\boldsymbol{\mathcal{R}}_t - \mathbf{c}||^2 \leq R^2 + \varepsilon_t, \varepsilon_t \geq 0,\ \forall t \end{aligned} \tag{12}$$

where $\mathbf{c}$ and R are the center and radius of the hypersphere respectively, $||\boldsymbol{\mathcal{R}}_t - c||^2$ is the distance between a graph representation and the center, ε_t is a slack variable introduced for $\boldsymbol{\mathcal{R}}_t$ to accommodate outliers during training, and C is a hyperparameter that balance the trade-off between the errors ε_t and the volume of the sphere. The objective defined in Eq. 12 aims to cluster all training samples within a minimum hypersphere using Lagrange multipliers, similar to SVDD [42]. We propose a multi-granularity loss function that considers both edge-level and graph-level objectives:

$$\mathcal{L} = \mathcal{L}_e + \alpha\mathcal{L}_g + \frac{\lambda}{2}\sum(||\mathbf{W}_g||_2^2 + ||\mathbf{W}_a||_2^2 + ||\mathbf{W}_1||_2^2 + ||\mathbf{W}_2||_2^2) \tag{13}$$

where $\mathbf{W}_a$ denotes the weights of temporal-attentive transformers. Hyperparameter α controls the trade-off between edge-level and graph-level violations, and λ modulates the weight decay L2 regularizer to avoid overfitting.

IV. Experiments

A. Experimental Settings

We evaluate our method in both the new anomalous relation detection setting and the traditional setting: (1) Edge-level detection: it aims at detecting anomalous relations in a log sequence, which are the edges in a log graph for a given time window. For each dataset, we label edges connected to the

TABLE II: Statistics of the three datasets.

	BGL	AIT	Sock Shop
# Log Messages	4,713,494	1,074,902	14,674
# Anomalies	348,460	45,651	408
# Nodes	4,393,108	1,663,188	10,340
Avg. degree	11.80	15.85	13.84
# Edges	25,918,022	13,180,752	71,540
# Anomalous edges	1,572,696	567,906	2,468
# Graphs	36,169	15,464	270
# Anomalous graphs	2,659	1,078	16

annotated anomalous logs as anomalies under this new setting. (2) Interval-level detection: it aims at detecting anomaly time windows which contains anomalous logs. We use this setting for a fair comparison with traditional log anomaly detection methods and more recent graph-based anomaly detection methods. In this setting, we treat a time window as anomaly if it contains any labeled anomalous logs. This is equivalent to the graph-level detection in our context.

Datasets. Among several potential candidates, we choose three publicly available datasets or platforms that have been examined by previous researches. We collect log sequences from these data sources to evaluate the effectiveness of our approach. Below we describe the details of the three datasets, and their statistics is shown in Table II.

- BlueGene/L (BGL) [57]. BGL is an open dataset of logs collected from a BlueGene/L supercomputer system with 131,072 processors and 32,768GB memory. The logs can be categorized into alert (anomalous) and non-alert (normal) messages identified by alert category tags.
- Austrian Institute of Technology (AIT) [58]. AIT (v1.1) is collected from four independent testbeds. Each of the web servers runs Debian and a set of installed services such as Apache2, PHP7, Exim4, Horde, and Suricata. Furthermore, the data includes logs from 11 Ubuntu hosts on which user behaviors were simulated.
- Sock Shop Microservices [59]. Sock Shop is a test bed that can be used to illustrate microservices architectures, demonstrate platforms at talks and meetups, or as a training and education tool. Specifically, we deploy and generate anomalous relations by adding shopping items that have not been browsed by each customer or introducing a large number of items in certain time periods.

Baselines. We compare the performance of GLAD with a wide range of baselines. For the edge-level setting, we consider five graph-based anomaly detection baselines. For fairness, except for AddGraph [41] that directly identifies anomalous edges, we use their reconstructed node feature vectors to compute edge scores and evaluate their edge-level performance.

- DOMINANT [32]. DOMINANT contains a GCN encoder, a structure reconstruction decoder and a attribute reconstruction decoder. It learns a weighted reconstruction errors as the node anomalous score.
- CONAD [35]. CONAD first generates augmented graphs based on prior human knowledge of anomaly types, then applies a Siamese GNN to detect node anomalies.
- AnomalyDAE [33]. AnomalyDAE uses a structure encoder-decoder to learn structure reconstruction errors and an attribute encoder-decoder to learn feature reconstruction errors. These errors are balanced to form an anomalous score of each node.
- MLPAE [60]. MLPAE applies a Multi-Layer Perceptron (MLP) autoencoder to detect anomalous nodes without considering structure information in a graph.
- AddGraph [41]. AddGraph incorporates a temporal-attentive RNN into a GCN encoder to learn structure and attribute representations in dynamic graphs. It learns edge scores according to pairwise node latent vectors and detects anomalous edges.

For the interval-level setting, existing works focusing on logs can be divided into two categories: 1) sequence-based methods, including traditional methods such as PCA [11], Isolation Forest [9], OCSVM [7] and deep learning-based methods such as DeepLog [6], LogAnomaly [61], LogBERT [12]; and 2) graph-based methods, including LogGD [62], LogFlash [63], and DeepTraLog [64].

- Principal Component Analysis (PCA) [11]. PCA builds a counting matrix according to the log event frequency and then maps the matrix into a latent space to detect anomalous sequences.
- Isolation Forest (iForest) [9]. An unsupervised learning method that represents features as tree structures for anomaly detection.
- One-class SVM (OCSVM) [7]. A well-known one-class classification method by building a feature matrix based on the norm data for anomaly detection.
- DeepLog [6]. DeepLog uses LSTM to capture patterns of normal log sequences and further identifies anomalous log sequences based on log key predictions.
- LogAnomaly [61]. LogAnomaly proposes template2vec to extract log template semantics and use LSTM to detect sequential and quantitative log anomalies.
- LogBERT [12]. LogBERT uses BERT to encode each log sequence into a feature space by self-supervision, and detect anomalous log sequences via hypersphere learning.
- LogGD [62]. LogGD constructs directed graph by connecting log templates following sequential relations, and identifies anomalies via graph classification based on Graph Transformer network.
- LogFlash [63]. LogFlash builds a time-weighted control flow graph (TCFG), where nodes are log templates and edges represent the transition between them, and compare log streams with TCFG to find deviations.
- DeepTraLog [64]. DeepTraLog constructs trace event graph (TEG) to represent various relations between the span/log events of the trace. It learns a gated GNN-based SVDD representation for each TEG and identifies anomalies via hypersphere learning.

To evaluate the impact of individual components in GLAD on the final performance, we also conduct experiments on different GLAD variants:

- GLAD$^{\xi}$. In this variant, GLAD applies a rule-based (instead of prompt-based) field extraction during graph

TABLE III: Edge-level performance (%) of GLAD and baseline methods. **Bold** numbers denote the best metric among all the methods. We ran each model 5 times to get the average results.

Method	BGL					AIT					Sock Shop				
	Precision	Recall	F-1	AUC	AUPR	Precision	Recall	F-1	AUC	AUPR	Precision	Recall	F-1	AUC	AUPR
DOMINANT	35.09	90.68	50.60	42.99	24.30	39.94	89.56	55.24	42.01	43.39	11.59	**95.63**	20.68	39.83	13.33
CONAD	32.19	**97.83**	48.44	44.84	25.99	39.93	89.54	55.23	44.48	45.47	16.31	88.92	27.56	43.82	17.49
AnomalyDAE	36.34	88.27	51.48	45.31	26.03	**55.12**	70.63	61.92	45.49	45.17	12.40	93.93	21.90	39.70	12.53
MLPAE	35.53	82.41	49.65	43.76	24.65	39.94	89.58	55.25	43.71	43.60	11.33	93.87	20.22	38.22	11.34
AddGraph	**48.21**	66.27	55.82	54.29	33.97	48.96	85.92	62.38	46.16	46.94	34.49	84.51	48.99	51.39	58.52
GLAD$^{\xi}$	38.94	74.26	51.09	50.73	30.06	44.91	89.87	59.89	45.16	46.30	35.79	80.14	49.48	50.74	52.88
GLAD$^{\wr}$	40.60	73.31	52.26	50.34	30.82	45.17	90.71	60.31	45.83	46.12	32.28	82.77	46.45	48.57	52.39
GLAD†	39.53	89.56	54.85	52.97	31.07	50.08	**93.30**	65.18	46.29	46.53	50.86	82.76	63.01	61.85	65.42
GLAD	47.09	86.06	**60.87**	**56.56**	**38.99**	54.15	90.81	**67.84**	**49.09**	**48.66**	**56.02**	91.00	**69.35**	**61.93**	**68.37**

configuration. This allows us to evaluate the significance of accurate identification of system entities and their interrelations with log events.

- GLAD$^{\wr}$. This version of GLAD is designed without the use of transformer encoder, removing its ability to capture temporal features. The purpose is to investigate the significance of temporal features.
- GLAD†. This variant removes multi-granularity learning from the training process of GLAD, thereby examining the importance of global features in detecting anomalies.

Metrics. We measure the model performance on anomaly detection based on three widely-used classification metrics, including Precision, Recall, and F-1 score, as well as two ranking metrics, including AUC and AUPR score.

Implementation Details. All GNN models in our research are built on PyTorch Geometric (PyG) framework. These models are configured with two layers, with input channels set at 768, and output channels at 1,024. For Sentence-BERT and BART, we use their pre-trained models, namely *bert-base-uncased* and *facebook/bart-base*, from Hugging Face. For field extraction, we either fine-tune BART over 100 epochs using 10-shot training samples or use pre-defined regular expressions. For anomaly detection, we split the log sequences into a ratio of 6:1:3, where 60% as training set, 10% as validation set, and 30% as test set. We apply an unsupervised learning paradigm [7], [61], [64] where only normal log sequences are used for training, and train each model for 100 epochs. Hyperparameters are adjusted via grid search on the validation set. Specifically, we use AdamW optimizer [65] with a learning rate of 1e-3, μ of 0.3, γ of 0.5, global weight α of 1, and weight decay λ of 5e-7. Our analysis operates with a window size of 60 seconds. Our work is conducted in a leading industry company using an NVIDIA RTX A4500 GPU. Our GLAD has been deployed to monitor internal cloud system log data for anomaly detection.

B. Experimental Results

Edge-level Performance. We first compare GLAD with baseline methods in terms of their edge-level performance. As shown in Table III, we observe that: (1) GLAD outperforms all baseline methods in F-1 score, AUC score and AUPR score. This demonstrates the efficacy of our approach in identifying anomalous relations between log fields and log events. (2) While some baseline methods like DOMINANT, CONAD, AddGraph in BGL dataset, and AnomalyDAE, MLPAE in Sock Shop dataset, achieve high recall scores, and AnomalyDAE in AIT dataset, AddGraph in BGL dataset achieve high precision scores, their F-1 scores are relatively low. This suggests that they either adopt an overly cautious stance towards anomalies or produce a high number of false positives by erroneously classifying many samples as anomalies. (3) Edge-level anomaly detection is notably more challenging compared to interval-level anomaly detection, e.g., no method in the three datasets achieved a precision score exceeding 60% or an F-1 score above 70%. (4) Those method optimized on edge-level, i.e., AddGraph and GLAD, achieve better precision, F-1, AUC and AUPR scores across all datasets. Specifically, they exhibit larger advantages over other methods in our generated Sock Shop datasets that contain specific anomalous relations, substantiating our hypothesis that edge-level learning can better detect anomalous relations that are elusive to other methods. (5) While some methods, such as AnomalyDAE, excel in one dataset, achieving a 61.92% F-1 score in the AIT dataset, they flounder in others, dropping to a 21.90% F-1 score in the Sock Shop dataset, for instance. (6) Compared to AddGraph, GLAD achieves superior performance, especially recall scores, across all three datasets. This demonstrates the advantages of our graph configuration and graph-based edge-level anomaly detection method.

TABLE IV: Interval-level performance (%) in the BGL dataset. We ran each model 5 times to get the average results.

Method	Precision	Recall	F-1	AUC	AUPR
PCA	9.04	**98.12**	16.56	55.64	9.03
iForest	**100.00**	14.74	25.70	57.37	21.64
OCSVM	1.09	12.48	2.00	28.22	7.32
DeepLog	89.02	80.54	84.57	89.26	70.17
LogAnomaly	91.40	79.32	84.93	92.98	75.21
LogBERT	91.47	92.69	92.07	96.33	82.70
LogGD	90.89	93.31	92.08	96.91	81.74
LogFlash	82.46	86.73	84.54	86.78	74.52
DeepTraLog	79.48	97.68	87.64	84.77	70.92
GLAD$^{\xi}$	88.35	89.86	89.10	95.63	80.44
GLAD$^{\wr}$	89.24	90.18	89.51	96.07	79.91
GLAD†	89.73	91.64	90.67	96.31	81.65
GLAD	90.82	94.57	**92.66**	**98.18**	**84.69**

Interval-level Performance. We further evaluate GLAD on the common interval-level protocol to demonstrate its effectiveness. Due to space limit, we only present the results of the widely used BGL dataset in Table IV. We observe that: (1) Compared to edge-level detection results, GLAD achieves much higher Precision (and F-1). Significantly, GLAD surpasses all methods with a leading F-1 of 92.66%, AUC

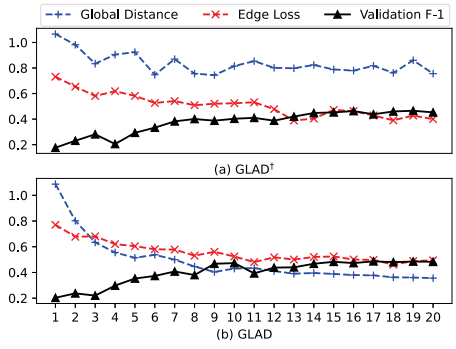


Fig. 5: Ablation study of multi-granularity learning.

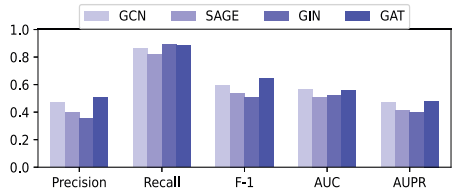


Fig. 6: GLAD performance using different GNN encoders.

of 98.18%, and AUPR of 84.69%. This demonstrates the proficiency of GLAD in capturing conventional anomalies in addition to relational ones. (2) Traditional sequence-based methods such as PCA, iForest, and OCSVM have significantly lower performance across all metrics. For instance, PCA suffers severely in Precision (9.04%), while iForest has a notably poor Recall (14.74%). OCSVM has the lowest performance among the three, with a negligible F-1 score of 2.00%. (3) DL sequence-based methods significantly outperform the traditional ones. Among these, LogBERT outperforms DeepLog and LogAnomaly with a F-1 score of 92.07% and an AUC of 96.33%. This showcases the effectiveness of transformers in capturing both long-term dependencies and semantic relations in log sequences. (4) Among graph-based methods, LogGD shows competitive results with an F-1 score of 92.08%, even slightly better than LogBERT. This suggests that the inclusion of both transformers and graph structures in hypersphere learning could benefit anomaly detection.

Ablation Study. With results of GLAD variants shown in Table III and Table IV, we observe that: (1) The performance gap between GLAD$^{\xi}$ and GLAD—roughly 9% F-1 improvement for edge-level detection in BGL—reveals the benefit of employing prompt-based filed extraction in graph configuration, thereby enhancing the effectiveness of GLAD in detecting anomalies. (2) The difference between GLAD$^{\wr}$ and GLAD underscores the significant role of temporal-attentive transformers. With the incorporation of temporal features, GLAD gains over 8% (and 3%) F-1 increases when detecting anomalous relations (and intervals) in BGL dataset. However, GLAD$^{\wr}$ still outperforms numerous baseline methods, asserting the robustness of our graph-based framework. (3) The comparison between GLAD† and GLAD indicates the positive impact of incorporating global features in anomaly detection, as evidenced by the superior F-1, AUC and AUPR scores of GLAD.

TABLE V: Two proposed prompts for field extraction.

Prompt $\mathbf{P}_1$
$\mathbf{P}^+$: $\langle candidate\_span \rangle$ is a/an $\langle entity\_type \rangle$ entity
$\mathbf{P}^-$: $\langle candidate\_span \rangle$ is not a named entity
Prompt $\mathbf{P}_2$
$\mathbf{P}^+$: $\langle entity\_type \rangle$ = $\langle candidate\_span \rangle$
$\mathbf{P}^-$: $\langle candidate\_span \rangle$ = none

TABLE VI: Performance of rule-based field extraction v.s. prompt-based n-shot field extraction.

Technique		Pre.	Rec.	F-1
regex		36.48	44.28	40.00
$\mathbf{P}_1$	1-shot	16.53	59.34	25.86
	5-shot	28.33	74.38	41.03
	10-shot	**66.28**	85.22	**74.57**
$\mathbf{P}_2$	1-shot	17.89	58.14	27.36
	5-shot	28.00	73.76	40.59
	10-shot	64.68	**87.82**	74.49

To further illustrate how multi-granularity learning benefits GLAD during training, we record the normalized global distance ($\mathcal{L}_g$ in Eq. 12), edge loss ($\mathcal{L}_e$ in Eq. 10) and validation F-1 scores after each training epoch in BGL dataset. In Figure 5, we observe that: (1) The comparison between GLAD† and GLAD in terms of global distance shows that hypersphere learning effectively clusters normal samples in the graph embedding space, i.e., the converged normalized global distance of GLAD is less than half of that of GLAD†. (2) The comparison between GLAD† and GLAD in terms of edge loss and validation F-1 score shows that hypersphere learning further improves the anomaly detection performance, i.e., the edge loss of GLAD decreases more stably and its validation F-1 outperforms that of GLAD† in the later training stage.

We also analyze the impact of using different GNN encoders, i.e., GCN [27], SAGE [29], GIN [28], GAT [30], on GLAD's performance in the BGL dataset. Figure 6 reveals that the performance of GLAD remains consistently robust across diverse GNN encoders, though some models excel in specific evaluation metrics, e.g., GCN and GIN show superior performance in terms of F-1, AUC, and AUPR scores. This resilience against changes in DL models demonstrates that GLAD can be flexibly deployed using various combinations of state-of-the-art architectures.

Field Extraction. To investigate the effectiveness of our field extraction method, we annotate n log messages with two prompts ($\mathbf{P}_1$, $\mathbf{P}_2$ in Table V) for each field type and train corresponding field extraction models. Note that we use $\mathbf{P}_1$ in GLAD for graph construction due to slightly better anomaly detection performance. As shown in Table VI, with only 5-shot learning, our field extraction model is as competitive as hand-crafted rules in terms of F-1 scores. Our method significantly outperforms rule-based method when conducting 10-shot learning, which explains the superiority of GLAD over GLAD$^{\xi}$ and suggests the practicability of our few-shot method in low-resource scenarios where annotations are limited.

Parameter Study. We investigate the impact of two critical hyperparameters—window size and the number of GNN

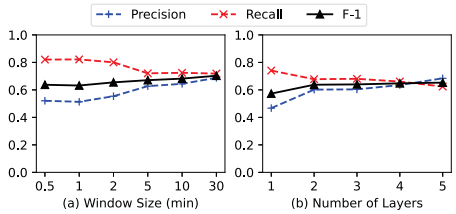


Fig. 7: GLAD performance with different parameters.

layers—on GLAD's performance in the BGL dataset. In each experiment, one hyperparameter is altered while the rest are held constant. As shown in Figure 7, the performance of GLAD, especially the F-1 scores, is robust to variations in these hyperparameters. This indicates that GLAD maintains its effectiveness across a range of configurations, highlighting its suitability for deployment in real-world scenarios. Specifically, Figure 7a suggests that a longer monitoring period leads to higher precision but lower recall. This implies that the window size can be tuned to balance between achieving higher true positive rates and reducing false positive rates. Similar strategy (Figure 7b) is applicable to configuration of GNN layers.

TABLE VII: Comparison of total training and testing overheads, and average overheads per log for different methods.

Method	Training Time		Testing Time	
	Total (s)	Avg. (ms)	Total (s)	Avg. (ms)
PCA	87.36	5	0.61	0
iForest	0.00	0	4.63	0
OCSVM	235.79	15	107.13	8
DeepLog	2321.21	155	1595.18	125
LogAnomaly	4420.02	296	2625.36	196
LogBERT	1950.31	130	470.14	37
LogGD	1753.82	117	1050.07	82
LogFlash	678.52	45	286.33	22
DeepTraLog	1261.55	84	796.81	62
GLAD	2490.40	166	1170.73	92

Efficiency Analysis. We compare the training and testing time of different methods in the BGL dataset. As shown in Table VII, traditional methods such as PCA, iForest, and OCSVM show small overheads as they are rather simple, among which OCSVM exhibits a rather high overhead due to construction of a feature matrix based on norm data for anomaly detection. DL sequence-based methods such as DeepLog, LogAnomaly, and LogBERT, generally possess higher overheads due to their complex architectures. Among them, LogAnomaly has the highest overhead due to complex template2vec learning process and low parallelism. While graph-based methods show rather lower overheads, underscoring the computational efficiency of graph structures. Interestingly, GLAD provides a training and testing overhead of 166 and 92 milliseconds per log, which is notably less than that of LogAnomaly and comparable to DeepLog. Given the sophisticated transformer and GNN architectures of GLAD, its relatively small overhead underscores our efficient design. This can be attributed to the direct application of temporal-attentive transformers on graph features, avoiding both tokenization and embedding, thereby increasing parallel computation.

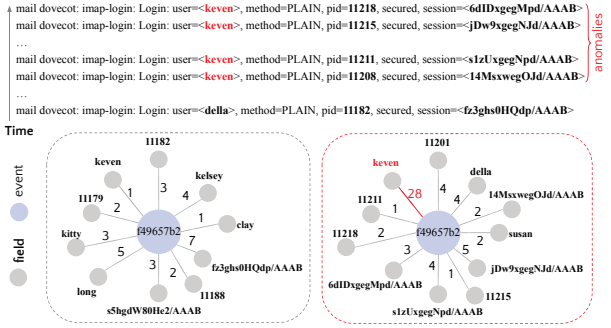


Fig. 8: Visualization of a normal graph (left) and anomalous graph (right). Edge marked in red denotes anomalous relation.

Case Study. To provide deeper insight into the performance of our graph-based anomaly detection, we visualize two sample graphs in Figure 8. In this case, all log messages share the same event template "f49657b2", and the anomalous relation manifests as the user "keven" making frequent requests to a server (28 times) compared to other users whose requests are considerably fewer. Existing methods that neglect system interactions cannot identify such anomalies, as they do not consider relations among system components. Our GLAD, however, successfully detects these anomalies by considering both the edge weight values and temporal patterns in a sequence of graphs. The comparison of the two constructed graphs also demonstrate the interpretability of our graph-based approach.

V. Conclusions

In this paper, we proposed a Graph-based Log Anomaly Detection framework, GLAD, which considers relational patterns in addition to log semantics and sequential patterns for system relation anomaly detection. First, a field extraction module utilizing prompt-based few-shot learning is used to extract field information, e.g., *service*, *user*, from log contents. Then, with the log events and fields extracted, dynamic log graphs can be constructed for sliding windows with events and fields as nodes, and the relations between them as edges. Finally, a temporal-attentive graph edge anomaly detection model is introduced for detecting anomalous relations from the dynamic log graphs, where a GNN-based encoder facilitated with transformers is used to model the structural, content, and temporal features. Experiments conducted on three datasets demonstrated the effectiveness of GLAD on system relation anomaly detection using system logs and providing deep insights into the anomalies.

References

[1] G. F. Jr., J. J. P. C. Rodrigues, L. F. Carvalho, J. Al-Muhtadi, and M. L. P. Jr., "A comprehensive survey on network anomaly detection," *Telecommun. Syst.*, no. 3, 2019.

[2] S. Budalakoti, A. N. Srivastava, and M. E. Otey, "Anomaly detection and diagnosis algorithms for discrete symbol sequences with applications to airline safety," *IEEE Trans. Syst. Man Cybern. Part C*, 2009.

[3] V. Chandola, A. Banerjee, and V. Kumar, "Anomaly detection for discrete sequences: A survey," *IEEE Trans. Knowl. Data Eng.*, 2012.

[4] B. Dong, Z. Chen, L. Tang, H. Chen, H. Wang, K. Zhang, Y. Lin, and Z. Li, "Anomalous event sequence detection," *IEEE Intell. Syst.*, 2021.

[5] B. Dong, Z. Chen, W. H. Wang, L. Tang, K. Zhang, Y. Lin, Z. Li, and H. Chen, "Efficient discovery of abnormal event sequences in enterprise security systems," in *CIKM*, 2017.

[6] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *CCS*, 2017.

[7] B. Schölkopf, J. C. Platt, J. Shawe-Taylor, A. J. Smola, and R. C. Williamson, "Estimating the support of a high-dimensional distribution," *Neural Comput.*, 2001.

[8] Q. Lin, H. Zhang, J. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *ICSE*, 2016.

[9] F. T. Liu, K. M. Ting, and Z. Zhou, "Isolation forest," in *ICDM*, 2008.

[10] J. Lou, Q. Fu, S. Yang, Y. Xu, and J. Li, "Mining invariants from console logs for system problem detection," in *USENIX*, 2010.

[11] W. Xu, L. Huang, A. Fox, D. A. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *ICML*, 2010.

[12] H. Guo, S. Yuan, and X. Wu, "Logbert: Log anomaly detection via BERT," in *IJCNN*, 2021.

[13] S. Boriah, V. Chandola, and V. Kumar, "Similarity measures for categorical data: A comparative evaluation," in *SDM*, 2008.

[14] D. J. Weller-Fahy, B. J. Borghetti, and A. A. Sodemann, "A survey of distance and similarity measures used within network intrusion anomaly detection," *IEEE Commun. Surv. Tutorials*, 2015.

[15] Z. Liu, N. Japkowicz, R. Wang, Y. Cai, D. Tang, and X. Cai, "A statistical pattern based feature extraction method on system call traces for anomaly detection," *Inf. Softw. Technol.*, 2020.

[16] S. Cho and H. Park, "Efficient anomaly detection by modeling privilege flows using hidden markov model," *Comput. Secur.*, 2003.

[17] M. J. Zhao, A. R. Driscoll, S. Sengupta, R. D. F. Jr., D. J. Spitzner, and W. H. Woodall, "Performance evaluation of social network anomaly detection using a moving window-based scan method," *Qual. Reliab. Eng. Int.*, 2018.

[18] P. Bodík, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *EuroSys*, 2010.

[19] R. Chen, S. Zhang, D. Li, Y. Zhang, F. Guo, W. Meng, D. Pei, Y. Zhang, X. Chen, and Y. Liu, "Logtransfer: Cross-system log anomaly detection for software systems with transfer learning," in *ISSRE*, 2020.

[20] Y. Liang, Y. Zhang, H. Xiong, and R. K. Sahoo, "Failure prediction in IBM bluegene/l event logs," in *ICDM*, 2007.

[21] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *ICSE*, 2021.

[22] M. M. Breunig, H. Kriegel, R. T. Ng, and J. Sander, "LOF: identifying density-based local outliers," in *SIGMOD*, 2000.

[23] 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 *KDD*, 2022.

[24] F. Wang, A. Bundy, X. Li, R. Zhu, K. Nuamah, L. Xu, S. Mauceri, and J. Z. Pan, "LEKG: A system for constructing knowledge graphs from log extraction," in *IJCKG*, 2021.

[25] A. Ekelhart, F. J. Ekaputra, and E. Kiesling, "The SLOGERT framework for automated log knowledge graph construction," in *ESWC*, 2021.

[26] K. Kurniawan, A. Ekelhart, E. Kiesling, D. Winkler, G. Quirchmayr, and A. M. Tjoa, "Virtual knowledge graphs for federated log analysis," in *ARES*, 2021.

[27] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," in *ICLR*, 2017.

[28] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *ICLR*, 2019.

[29] W. L. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *NeurIPS*, 2017.

[30] P. Velickovic, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," in *ICLR*, 2018.

[31] Y. Shi, Z. Huang, S. Feng, H. Zhong, W. Wang, and Y. Sun, "Masked label prediction: Unified message passing model for semi-supervised classification," in *IJCAI*, 2021.

[32] K. Ding, J. Li, R. Bhanushali, and H. Liu, "Deep anomaly detection on attributed networks," in *SDM*, 2019.

[33] H. Fan, F. Zhang, and Z. Li, "Anomalydae: Dual autoencoder for anomaly detection on attributed networks," in *ICASSP*, 2020.

[34] Z. Chen, B. Liu, M. Wang, P. Dai, J. Lv, and L. Bo, "Generative adversarial attributed network anomaly detection," in *CIKM*, 2020.

[35] Z. Xu, X. Huang, Y. Zhao, Y. Dong, and J. Li, "Contrastive attributed network anomaly detection with data augmentation," in *PAKDD*, 2022.

[36] W. Yu, W. Cheng, C. C. Aggarwal, K. Zhang, H. Chen, and W. Wang, "Netwalk: A flexible deep embedding approach for anomaly detection in dynamic networks," in *KDD*, 2018.

[37] P. Zheng, S. Yuan, X. Wu, J. Li, and A. Lu, "AAAI," 2019.

[38] L. Ouyang, Y. Zhang, and Y. Wang, "IJCNN," 2020.

[39] D. Duan, L. Tong, Y. Li, J. Lu, L. Shi, and C. Zhang, "AANE: anomaly aware network embedding for anomalous link detection," in *ICDM*, 2020.

[40] L. Cai, Z. Chen, C. Luo, J. Gui, J. Ni, D. Li, and H. Chen, "Structural temporal graph neural networks for anomaly detection in dynamic graphs," in *CIKM*, 2021.

[41] L. Zheng, Z. Li, J. Li, Z. Li, and J. Gao, "Addgraph: Anomaly detection in dynamic graph using attention-based temporal GCN," in *IJCAI*, 2019.

[42] L. Ruff, N. Görnitz, L. Deecke, S. A. Siddiqui, R. A. Vandermeulen, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *ICML*, 2018.

[43] X. Teng, M. Yan, A. M. Ertugrul, and Y. Lin, "Deep into hypersphere: Robust and unsupervised anomaly discovery in dynamic networks," in *IJCAI*, 2018.

[44] M. Zheng, C. Zhou, J. Wu, S. Pan, J. Shi, and L. Guo, "Fraudne: a joint embedding approach for fraud detection," in *IJCNN*, 2018.

[45] H. Wang, C. Zhou, J. Wu, W. Dang, X. Zhu, and J. Wang, "Deep structure learning for fraud detection," in *ICDM*, 2018.

[46] Y. Dou, K. Shu, C. Xia, P. S. Yu, and L. Sun, "User preference-aware fake news detection," in *SIGIR*, 2021.

[47] N. Reimers and I. Gurevych, "Sentence-BERT: Sentence embeddings using Siamese BERT-networks," in *(EMNLP-IJCNLP)*, 2019.

[48] H. Hamooni, B. Debnath, J. Xu, H. Zhang, G. Jiang, and A. Mueen, "Logmine: Fast pattern recognition for log analytics," in *CIKM*, 2016.

[49] S. Messaoudi, A. Panichella, D. Bianculli, L. C. Briand, and R. Sasnauskas, "A search-based approach for accurate identification of log message formats," in *ICPC*, 2018.

[50] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," in *ACL*, 2020.

[51] L. Cui, Y. Wu, J. Liu, S. Yang, and Y. Zhang, "Template-based named entity recognition using BART," in *Findings of ACL/IJCNLP*, 2021.

[52] Y. Li, X. Yu, Y. Liu, H. Chen, and C. Liu, "Uncertainty-aware bootstrap learning for joint extraction on distantly-supervised data," in *ACL*, 2023.

[53] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NeurIPS*, 2017.

[54] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *ICWS*, 2017.

[55] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes," in *AAAI*, 2014.

[56] A. Bordes, N. Usunier, A. García-Durán, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *NeurIPS*, 2013.

[57] A. J. Oliner and J. Stearley, "What supercomputers say: A study of five system logs," in *DSN*, 2007.

[58] M. Landauer, F. Skopik, M. Wurzenberger, W. Hotwagner, and A. Rauber, "Have it your way: Generating customized log datasets with a model-driven simulation testbed," *IEEE Transactions on Reliability*, 2021.

[59] A. Giurgiu and I. Crosby, "Sock Shop Microservices Demo," 12 2017. [Online]. Available: https://github.com/microservices-demo

[60] M. Sakurada and T. Yairi, "Anomaly detection using autoencoders with nonlinear dimensionality reduction," in *MLSDA*, 2014.

[61] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun, and R. Zhou, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *IJCAI*, 2019.

[62] Y. Xie, H. Zhang, and M. A. Babar, "Loggd: Detecting anomalies from system logs with graph neural networks," in *QRS*, 2022.

[63] 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 *ISSRE*, 2021.

[64] C. Zhang, X. Peng, C. Sha, K. Zhang, Z. Fu, X. Wu, Q. Lin, and D. Zhang, "Deeptralog: Trace-log combined microservice anomaly detection through graph-based deep learning," in *ICSE*, 2022.

[65] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," in *ICLR*, 2019.