DEBUGGING TABULAR LOG AS DYNAMIC GRAPHS

Anonymous authors
Paper under double-blind review

ABSTRACT

Tabular log abstracts objects and events in the real-world system and reports their updates to reflect the change of the system, where one can detect real-world inconsistencies efficiently by debugging corresponding log entries. However, recent advances in processing text-enriched tabular log data overly depend on large language models (LLMs) and other heavy-load models, thus suffering from limited flexibility and scalability. This paper proposes a new framework, GraphLogDebugger, to debug tabular log based on dynamic graphs. By constructing heterogeneous nodes for objects and events and connecting node-wise edges, the framework recovers the system behind the tabular log as an evolving dynamic graph. With the help of our dynamic graph modeling, a simple dynamic Graph Neural Network (GNN) is representative enough to outperform LLMs in debugging tabular log, which is validated by experimental results on real-world log datasets of computer systems and academic papers.

1 INTRODUCTION

Tabular log data plays a crucial role in representing and tracking the state and evolution of real-world systems. These logs are structured as rows of log entries, each capturing an event involving certain objects and their attributes at a specific time point. Common examples include system logs recording computing services (Zhu et al., 2023a), research logs tracking scientific publication activities (Clement et al., 2019), and interaction logs from multi-agent systems powered by large language models (LLMs) (Zhang et al., 2025b). Debugging of tabular logs is essential: it allows practitioners to detect anomalies in the original systems through efficient inspection of associated log records.

Log anomaly detection (He et al., 2016) has therefore been a long-standing research field in different niche areas, where data distributions are invariant or have little change. Existing frameworks (Du et al., 2017; Meng et al., 2019; Zhang et al., 2019; Pei et al., 2020; Guo et al., 2021; Chen & Tsourakakis, 2022) benefit from manually defined data structures or templates for log parsing which are often tailored to certain domains and thus yield absolute success in specific areas like computer system log or financial event log. However, due to this domain-specific principle, designing a general-purpose log debugger always remains challenging.

Efforts to overcome this challenge have led to two main lines of work, as shown in Figure 1. One stream focuses on graph modeling of the log data (Cheng et al., 2020; Zehra et al., 2021; Pang et al., 2025), where information in tabular log is gathered in a unified data structure: the graph, such as constructing knowledge graphs or text-rich dynamic graphs for computer system log (Sui et al., 2023; Li et al., 2023). Although these methods are both efficient and powerful, many of them lack flexibility: they still customize static graph structures for certain domains. Another stream explores LLM-based solutions, such as LLM prompting (Yu et al., 2023; Qi et al., 2023; Park, 2024) or retrieval-augmented generation (RAG) (Pan et al., 2024; Zhang et al., 2025a; Wang et al., 2025) pipelines. While these methods demonstrate general capabilities in text-based reasoning, thus showing potential of generalization, they often come with significant drawbacks: high computational costs, slow inference, and difficulty scaling to long log streams or resource-constrained settings.

Inspired by the idea to unify multimodal information in dynamic graphs (Feng et al., 2025), we propose **GraphLogDebugger**, a general and efficient framework for debugging tabular logs through dynamic graph modeling. Our core idea is to interpret tabular log entries as the evolving state of a hidden system, which can be reconstructed as a dynamic heterogeneous graph. We treat objects and events as different types of nodes with text embeddings empowered by modern language embed-

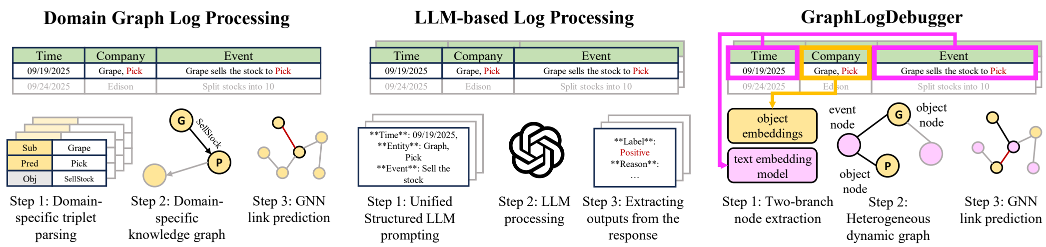

Figure 1: **Comparing GraphLogDebugger with two existing lines of works**. Processing log with domain-specific graphs requires custom text parsing, which lacks flexibility. LLM-based log processing overcomes this shortcoming by the general comprehension skills of LLMs, but suffers from poor efficiency. GraphLogDebugger combines the advantages of graph representation and those of LLMs and balances well generalizability and scalability.

ding models, and use the tabular structure to generate time-stamped connections between them. As new log entries arrive, they incrementally update the dynamic graph, capturing both structural and temporal dependencies. This formulation allows us to apply a lightweight dynamic Graph Neural Network (GNN) to perform online anomaly detection by evaluating the likelihood of new connections. Our approach avoids reliance on heavy LLMs while still capturing rich semantic and relational information in the data. Experimental results on real-world datasets from computer system logs and scientific publication logs validate the effectiveness of our approach. Despite its simplicity, our dynamic GNN framework outperforms LLM-based baselines in both accuracy and efficiency, demonstrating that dynamic graph modeling is a highly expressive yet lightweight alternative. Our contributions can be summarized as follows:

- We introduce a novel view of tabular logs as dynamic heterogeneous graphs, bridging the gap between structured attributes and semantic reasoning, and redefine the framework of online log anomaly detection, where object-event connections in each incoming log are evaluated through link prediction on the evolving graph.
- We propose a lightweight GNN-based debugger that can efficiently and accurately detect anomalies without using LLMs, and validate its performance on real-world datasets with diverse modalities.

2 Related Works

Tabular Log Processing. Many real-world logs include structured, time-stamped tabular attributes alongside annotated text fields. Examples come from financial prices paired with event series (Tetlock, 2007; Ruiz et al., 2012; Dong et al., 2024), scientific publication metadata (Clement et al., 2019; Kinney et al., 2023), healthcare records (Johnson et al., 2023), computer system logs (Zhu et al., 2023a), and multi-agent system reports (Zhang et al., 2025b). A key challenge in processing tabular logs with machine learning lies in capturing multi-attribute correlations while maintaining comprehension of their semantics (Wu et al., 2025). One common approach integrates main attributes recognized by human priors into structured data (Yang et al., 2018; Zhao & Feng, 2022; Koval et al., 2024), and then subsequently augments the representation by retrieval (Kurisinkel et al., 2024; Xiao et al., 2025). This modeling achieves good performance in domain-specific data, but lacks flexibility and generalizability for adaptation to other fields (Gardner et al., 2024).

An emerging alternative leverages Large Language Models (LLMs) (Brown et al., 2020), which have demonstrated strong generalizability in understanding, predicting, and generating tabular data (Liu et al., 2023; Zhang et al., 2024b; Fang et al., 2024; Wang et al., 2024b). By parsing diverse logs into a unified format with LLMs (Zhong et al., 2024), these models can be applied to downstream tasks that require reasoning capabilities, such as predicting stock prices (Yu et al., 2023), electricity demand (Wang et al., 2024a), and future events (Shi et al., 2023; Ye et al., 2024). However, LLM-based approaches often suffer from high overheads, complex deployment, and limited throughput. There remains a strong need for lighter-weight alternatives with comparable performance.

Dynamic Graphs. Graph Neural Networks (GNNs) have become a foundational paradigm for learning on graph-structured data (Kipf, 2016; Hamilton et al., 2017; Gilmer et al., 2017). Static

GNN models have benefited from advances in message passing (Battaglia et al., 2018), architectural depth (Li et al., 2021; Dwivedi et al., 2020), and inductive scalability (Hamilton et al., 2017). However, many real-world systems are dynamic, motivating models that capture both structural and temporal dependencies. Early approaches used recurrent layers or time-aware embeddings (Li et al., 2017; Seo et al., 2018) to extend static GNNs to dynamic settings (Pareja et al., 2020; Sankar et al., 2020; Kumar et al., 2019). Recent methods have embraced memory modules (Rossi et al., 2020) and temporal encoding (Xu et al., 2020) for finer-grained modeling of time-stamped interactions. Building on this trajectory, ROLAND (You et al., 2022) offers a framework that adapts static GNNs to dynamic graphs via hierarchical state propagation and live-update evaluation, which inspires new advances in benchmarks (Longa et al., 2023; Huang et al., 2023; Zhang et al., 2024a), architectures (Zhu et al., 2023b), explainability (Chen & Ying, 2023), and robustness (Zhang et al., 2023b).

Log Anomaly Detection. Log-based anomaly detection has long been a critical task for system reliability, and early neural approaches typically rely on sequence modeling via LSTMs (Du et al., 2017), CNNs (Lu et al., 2018), and autoencoders (Zhang et al., 2021; Castillo et al., 2022; Zhang et al., 2023a). Others incorporate adversarial training (Duan et al., 2021; He et al., 2023), or temporal networks (Zhang et al., 2019; Yang et al., 2021). More recently, pretrained language models have been adopted for log anomaly detection, either via fine-tuning (Guo et al., 2021; Lee et al., 2023) or prompt-based pipelines (Qi et al., 2023; Liu et al., 2024). Retrieval-augmented (No et al., 2024; Pan et al., 2024; Zhang et al., 2025a) methods have further pushed semantic understanding in LLM-based methods. As mentioned, while machine learning-based methods are highly domain-specific, LLM-based methods show some generalizability at a high cost.

One potential solution towards general and scalable methods for log debugging is to introduce dynamic graphs, where tabular log is considered as an evolving system and maintained in a dynamic graph. Early exploration makes use of knowledge graphs (Hogan et al., 2021) with domain specific parsing to generate triplets (Cheng et al., 2020; Zehra et al., 2021; Sui et al., 2023). Recent advances adopt dynamic graphs with text-rich nodes to represent tabular log (Li et al., 2023; Pang et al., 2025). Nevertheless, these works are either domain specific or LLM-based, yet not escaping from the dilemma between generalizability and scalability.

3 Preliminaries

Tabular log is the data modality used to report the update of real-world systems from the perspective of states and relations. It can be formally defined by a time series $X = \{x_0, x_1, x_2, ..., x_{N-1}\}$ annotated by a timestamps sequence $t_0 < t_1 < t_2 < ... < t_{N-1}$, where each of x_n is a log entry that contains different attributes x_n^m in the table: $x_n = \{x_n^0, x_n^1, x_n^2, ..., x_n^{M-1}\}$. Summarizing the general case of tabular log data in finance (Dong et al., 2024), healthcare (Johnson et al., 2023), academics (Clement et al., 2019), and other systems (Zhu et al., 2023a), we can separate attributes in the tabular log into three types:

- **Object:** Attributes that represent stand-alone objects in the tabular log, such as companies in the financial news log and cities in the medical record log.
- **Event:** Attributes that describe an event with text, for example, news content in the financial news log and record content in the medical record log. These attributes are usually the center of log entries, where other attributes supplement details and involved objects of the event. Without loss of generality, one log entry only has one Event attribute, because we could merge the text sections of different event attributes into one.
- **Feature:** Attributes that describe features related to the event or objects. For instance, the age is a feature of the patient object in the medical record log. Timestamp t_n is a special type of feature that provides the details about the time of the event.

In practice, we find that Objects and Features are mutually convertible. For example, the address of the company could be either an independent object or a feature of the company object in the financial log. Hence, the arrangement of Objects and Features is a hyperparameter that needs pre-definition.

While tabular log abstracts the change of real-world systems, it is expected that we could detect inconsistencies of the system from the corresponding tabular log. Based on the above categorization, we could then define three types of anomalies and corresponding anomaly detection tasks in the tabular log. Give a log entry $x_n = \{x_n^0, x_n^1, x_n^2, ..., x_n^{M-1}\}$ in the tabular log X:

- **Object anomaly:** Let $\{o_n^0, o_n^1, o_n^2, ..., o_n^{P-1}\}(P < M)$ be the object set. We have label $y_n = \{y_n^0, y_n^1, y_n^2, ..., y_n^{P-1}\}$, where $y_n^m = 0$ means that o_n^p is a normal object and $y_n^m = 1$ means that o_n^p is an abnormal object for the log entry x_n.
- **Event anomaly:** Let s_n be the event. We have a label y_n, where $y_n = 0$ means that s_n is a normal event and $y_n = 1$ means that s_n is an abnormal event for the log entry x_n.
- **Feature anomaly:** Let $\{f_n^0, f_n^1, f_n^2, ..., f_n^{Q-1}\}(Q < M)$ be the feature set. We have label $y_n = \{y_n^0, y_n^1, y_n^2, ..., y_n^{Q-1}\}$, where $y_n^q = 0$ means that o_n^q is a normal feature and $y_n^q = 1$ means that o_n^q is an abnormal feature for the log entry x_n.

Considering the tabular log X as an online system where new log entries come dynamically in time order, we could then define the anomaly detection in tabular log as an online anomaly detection task:

Definition 3.1 (Online Anomaly Detection of Tabular Log). Given an online system that dynamically produces log entry x_n, online anomaly detection for tabular log predicts its anomaly label y_n based on historical log entries $X_n = \{x_0, x_1, x_2, ..., x_{n-1}\}$

Notably, object anomaly and feature anomaly are isomorphic. Considering the fact that objects and features are convertible, the rest of this paper only studies object anomaly and event anomaly. Table 3 summarizes all used variables.

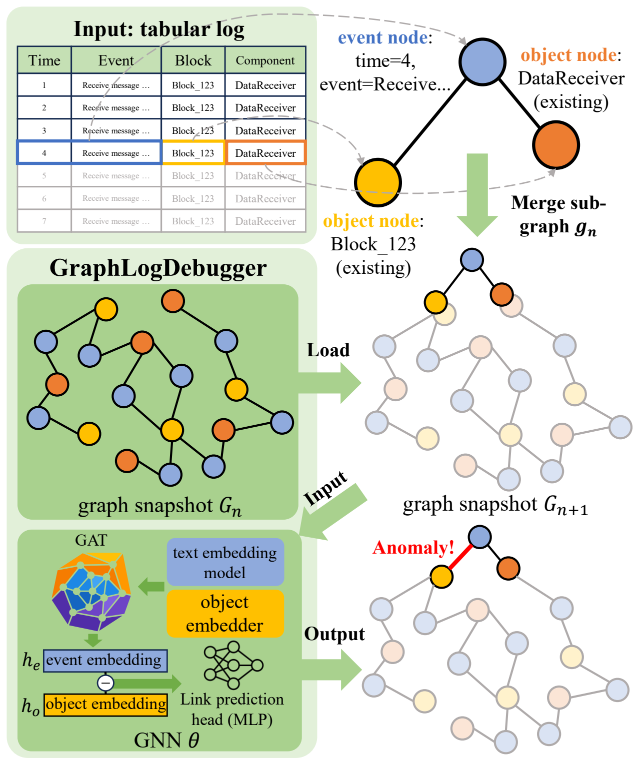

Figure 2: **GraphLogDebugger framework**. The framework checkpoints the GNN θ and the dynamic graph snapshot G_n. When a new log entry emerges, we first extract a sub-graph g_n and use it to update the dynamic graph. Then, we predict the links introduced by g_n in the dynamic graph by GNN θ, whose results indicate the anomaly.

4 GRAPHLOGDEBUGGER

We first integrate tabular log to a heterogeneous dynamic graph (Section 4.1). Then, we reformulate online anomaly detection of tabular log as dynamic graph anomaly detection(Section 4.2). Finally, we apply a dynamic GNN to debug the tabular log (Section 4.3).

4.1 INTEGRATING ONLINE TABULAR LOG TO DYNAMIC GRAPHS

Objects and events in the same log entry are naturally connected in the tabular log, from which we could construct graphs. To this end, we first define the graph structure within one log entry. As shown in Figure 2 (upper section), we build nodes v for both objects and the unique event in the new log entry. Each event and all its objects are connected by an edge e. This yields a sub-graph g_n for each log entry x_n.

Figure 2 (middle section) illustrates the composition of a dynamic graph $\mathcal{G}$ that stores the information of all historical log entries. This dynamic graph gathers all sub-graphs of log entries. We merge identical object nodes so that these sub-graphs are connected. Note that every event node should be unique. Every time a new log entry x_n emerges, we construct a sub-graph g_n accordingly and merge it into the dynamic graph $\mathcal{G}$. We denote the snapshot of $\mathcal{G}$ at time point t_n by G_n.

Algorithm 1: GraphLogDebugger: Online training for dynamic-graph anomaly detection

Input : Training log $X_{\text{train}} = \{x_{t_0}, \ldots, x_{t_{K-1}}\}$; GNN θ; text embedding model $\mathcal{F}$; negative sampling ratio ρ; threshold τ;
Output: Trained parameters $\theta^\star$, dynamic graph snapshot G_{t_K}

1 Initialize $\mathcal{G} : \mathcal{V} = \emptyset, \mathcal{E} = \emptyset$
2 **for** $k = 0, \ldots, K-1$ **do**
    // Integrate the incoming log entry
3   Build sub-graph g_k: creating nodes for object set $\mathcal{V}_k^o$ and event set $\mathcal{V}_k^e$ in x_{t_k} and connecting object-event pairs in x_{t_k} with edges
4   $\mathcal{G} \leftarrow (\mathcal{V} \cup \mathcal{V}_k^o, \mathcal{E})$
5   Positive set $\mathcal{E}_k^+ \leftarrow$ object-event links in g_k.
6   Negatives $\mathcal{E}_k^-$ by drawing $\rho \cdot |\mathcal{E}_k^+|$ non-existent object–event pairs in $\mathcal{G}$.
    // Embed nodes
7   Compute object embeddings h_o with GNN θ: $h_o = f_\theta(\mathcal{G})$
8   Compute new event embeddings h_e with text embedding model $\mathcal{F}$: $h_e = \mathcal{F}(\mathcal{V}_k^e)$
    // Predict links & Compute the loss
9   For each pair $(o, e) \in \mathcal{E}_k^+ \cup \mathcal{E}_k^-$, compute score $s_{o,e} = \sigma(\text{MLP}(\text{reduce}(h_o,\ h_e)))$.
10 Compute balanced BCE loss $\mathcal{L}_k$ on labels (1 for $\mathcal{E}_k^+$, 0 for $\mathcal{E}_k^-$) and update $\theta \leftarrow \theta - \eta \nabla_\theta \mathcal{L}_k$.
    // Updating the dynamic graph
11 $\mathcal{G} \leftarrow (\mathcal{V} \cup \mathcal{V}_k^e, \mathcal{E})$
12 Repeat Step 1-10 for epochs
13 **return** θ

Algorithm 2: GraphLogDebugger: Online evaluation for dynamic-graph anomaly detection

Input : Test log $X_{\text{test}} = \{x_{t_K}, \ldots, x_{t_{N-1}}\}$; trained GNN $\theta^\star$; last snapshot G_{t_K}; text embedding model $\mathcal{F}$; threshold τ
Output: Per-time link predictions $\{\mathcal{R}_{t_n}\}_{n=K}^{N-1}$; updated snapshot G_{t_N}

1 Initialize $\mathcal{G} \leftarrow G_{t_K}$ and load $\theta^\star$
2 **for** $n = K, \ldots, N-1$ **do**
    // Integrate the incoming log entry
3   Build sub-graph g_n from x_n with object set $\mathcal{V}_n^o$, new event set $\mathcal{V}_n^e$, and observed links $\mathcal{E}_n^+$
4   $\mathcal{G} \leftarrow (\mathcal{V} \cup \mathcal{V}_k^o, \mathcal{E})$
    // Embed nodes
5   Compute object embeddings on current snapshot: $h_o = f_{\theta^\star}(\mathcal{G})$
6   Compute embeddings for new events (time-aware): $h_e = \mathcal{F}(\mathcal{V}_n^e)$
    // Predict links
7   For each $(o, e) \in \mathcal{E}_n^+$, compute $s_{o,e} = \sigma(\text{MLP}(\text{reduce}(h_o,\ h_e)))$ and set $\hat{\ell}_{o,e} = \mathbb{1}[s_{o,e} \geq \tau]$
8   **Link prediction results:** $\mathcal{R}_{t_n} \leftarrow \{(o, e, s_{o,e}, \hat{\ell}_{o,e}) \mid (o, e) \in \mathcal{E}_n^+\}$
    // Update the dynamic graph
9   Accepted links $\hat{\mathcal{E}}_n^+ = \{(o, e) \in \mathcal{E}_n^+ \mid \hat{\ell}_{o,e} = 1\}$
10 Accepted new events $\hat{\mathcal{V}}_n^e = \{e \in \mathcal{V}_n^e \mid \exists o : (o, e) \in \hat{\mathcal{E}}_n^+\}$.
11 $\mathcal{G} \leftarrow (\ \mathcal{V} \cup \hat{\mathcal{V}}_n^e,\ \mathcal{E} \cup \hat{\mathcal{E}}_n^+\ )$
12 **return** $\{\mathcal{R}_{t_n}\}_{n=K}^{N-1}$ *and* $G_{t_N} = \mathcal{G}$

4.2 Debugging Tabular Log Graphs as Dynamic Graphs

Following the above integration, we transfer the online anomaly detection of tabular log defined in Section 3 into an anomaly detection problem of dynamic graph $\mathcal{G}$ (Ekle & Eberle, 2024):

- **Object anomaly detection:** The object anomaly occurs when the edge e between an object node and an event node is abnormal in the latest sub-graph g_n. This anomaly could then be detected by link prediction in the dynamic graph $\mathcal{G}$ (You et al., 2022), where a GNN is applied to predict the likelihood of e. If the likelihood exceeds a threshold, we consider the edge as normal. Otherwise, we consider the edge as an anomaly.
- **Event anomaly detection:** The event anomaly occurs when an abnormal event s is placed in the wrong entry in the latest sub-graph g_n. This means that all edges between this event are anomalies. We can therefore apply link prediction to all edges in the sub-graph and threshold the overall predicted likelihoods to determine the anomaly label of the event.

In a nutshell, the goal of our anomaly detection in Section 3 is equivalent to predicting the likelihood of all edges in the latest log entry sub-graph g_n, based on the dynamic graph snapshot G_n that stores all historical log entries of tabular log X_n. We finally transfer the online anomaly detection of tabular log into an anomaly detection problem in dynamic graphs, with notations omitted to Table 3:

Definition 4.1 (**Online Anomaly Detection of Tabular Log (Dynamic Graph)**). Given the snapshot G_n of a dynamic graph $\mathcal{G} = \{G_n\}_{n=0}^{N-1}$ and the new coming sub-graph $g_n = G_{n+1} \setminus G_n$, the goal is to predict the label y_n of links in g_n.

Table 1: **Statistics and details of the four datasets for tabular log debugging.**

Dataset	Domain	#Entries	#Objects	Event Attr.	Obj Attr.	Anomaly Type
Arxiv	Sci. Pub.	20,000	17316	title	authors	Event/Object
HDFS	System Log	20,000	2150	Content	Component,EventId,BlockId	Object
Analyst	Finance	20,000	3901	headline	publisher	Event
Landslide	Geology	20,000	8565	description	title, category,trigger,country	Object

4.3 Designing the GNN for dynamic graph anomaly detection

Figure 2 (bottom section) demonstrates the basic process of using our GNN to predict link anomaly labels. For details, our GNN θ takes the dynamic graph snapshot G_n and the incoming sub-graph g_n as inputs and predicts the likelihood of all links in g_n. The GNN consists of three parts: the node embedder, the GNN backbone, and the prediction head. First, the node embedder offers heterogeneous embeddings for all objects in G_n and g_n and event nodes in G_n. We exclude new events in g_n because we do not expect the outputs of the model will be interfered with by the graph structure in g_n. We assign a unique learnable embedding for each object, and use a pre-trained text embedding model to embed existing events. We also concatenate a time embedding to the event embedding based on the coming time t_n for event s. Our GNN backbone is adapted from the graph attention network (GAT) (Veličković et al., 2017), where we use two separate MLPs to map objects and events to the same space and apply GAT layers for message passing. The prediction head predicts the link between all object-event pairs in g_n. We first compare object embeddings after GAT layers and event text embeddings by reduction. We then pass the result to an MLP with Sigmoid activation to get the likelihood. We omit more details in the design space of the model in Appendix A.3.

Our GNN is trained under the setting of unsupervised anomaly detection (Pang et al., 2021): We separate the dataset into a training split and a test split by chronological order. In the training stage, all links in g_n are normal, and we provide negative examples for training by randomly sampling object-event pairs that are not connected. We append these fraud links to the ground-truth links to balance the label distribution and use them to train the GNN. After backpropagation, we finally update the dynamic graph snapshot G_n with subgraph g_n. Alg 1 summarizes the training algorithm.

In the test stage, we first resume the GNN as well as the latest dynamic graph snapshot G_n. This achieves the warm start of our debugger system. Then, we construct sub-graphs from the coming log entries and take them as parts of the evolving dynamic graph $\mathcal{G}$ that we succeed from the pre-trained GNN. The evaluation process is summarized in Alg 2.

5 Experiments

5.1 Experimental settings

Datasets. Our work provides a general framework of debugging different types of tabular log under the online setting. To validate this point, we span our experiments over datasets covering four different fields: (1) **Arxiv**: Tabular log recording the timestamps (from 2007-2025), the title and the authors of machine learning papers from the Arxiv (Clement et al., 2019) API; (2) **HDFS**: system log of Hadoop Distributed File System designed to run on commodity hardware (Xu et al., 2009; Zhu et al., 2023a), including the event content together with the objects related to the event; (3) **Analyst**: the commentary records on the finance by analysts, including title, author, and other features of posts [1]; (4) **Landslide**: event catalog reporting the global landslide [2]. These datasets contain both text-rich attributes and categorical attributes with diversified semantics, thus being challenging to process in one framework efficiently. Table 1 demonstrates the basic statistics of our four datasets.

We limit the maximum length of all tabular logs to 20,000 by slicing the original datasets. This is because LLM-based baseline methods are costly and not scalable, as discussed in the introduction. To ensure randomness, we randomly pick slices with a length of 20,000 from the whole sliced dataset. For datasets with multiple object attributes, we evaluate object anomaly detection, while for those with only one object attribute, we evaluate event anomaly detection, where event and object anomaly detection are equivalent. We summarize the basic setting of our four datasets in Table 1.

[1] `www.kaggle.com/datasets/miguelaenlle/massive-stock-news-analysis-db-for-nlpbacktests`

[2] `https://catalog.data.gov/dataset/global-landslide-catalog-export`

Table 2: **Our proposed GraphLogDebugger outperforms representative baselines on detection effectiveness and efficiency across diverse methods and datasets.** Higher is better for detection effectiveness; lower GFLOPs and higher Throughput are preferred for efficiency. "*" suggests some baselines always predict non-anomaly cases, leading to a 0 prediction, recall, and F1 score.

Dataset: **Arxiv** Task: **Event Anomaly**

Method	Detection Effectiveness				Efficiency	
	Acc.	Prec.	Recall	F1	GFLOPs	Throughput (it/s)
MLP	0.570 ± 0.205	0.556 ± 0.189	0.893 ± 0.331	0.676 ± 0.028	11.35	825.0 ±1429.0
RAG (Llama3-70b,k=5)	0.408 ± 0.038	0.426 ± 0.026	0.527 ± 0.029	0.471 ± 0.019	$\sim 10^5$	0.204 ± 0.009
RAG (GPT-oss-20b,k=5)	0.770 ± 0.106	0.771 ± 0.157	0.773 ± 0.014	0.772 ± 0.085	$\sim 10^4$	0.145 ± 0.018
RAG (Llama3-70b,k=10)	0.377 ± 0.014	0.400 ± 0.004	0.493 ± 0.038	0.442 ± 0.014	$\sim 10^5$	0.204 ± 0.015
RAG (GPT-oss-20b,k=10)	0.803 ± 0.090	0.798 ± 0.099	0.813 ± 0.087	0.805 ± 0.087	$\sim 10^4$	0.149 ± 0.014
GraphLogDebugger (Ours)	**0.957** ± 0.040	**0.920** ± 0.069	**1.000** ± 0.000	**0.959** ± 0.037	40.39	627.662 ± 7.623

Dataset: **Arxiv** Task: **Object Anomaly**

Method	Detection Effectiveness				Efficiency	
	Acc.	Prec.	Recall	F1	GFLOPs	Throughput (it/s)
MLP	0.570 ± 0.000	0.538 ± 0.000	0.990 ± 0.000	0.697 ± 0.000	11.35	552.0 ± 167.0
RAG (Llama3-70b,k=5)	0.455 ± 0.033	0.468 ± 0.026	0.667 ± 0.100	0.550 ± 0.052	$\sim 10^5$	0.208 ± 0.018
RAG (GPT-oss-20b,k=5)	0.597 ± 0.019	0.564 ± 0.016	0.850 ± 0.025	0.678 ± 0.004	$\sim 10^4$	0.039 ± 0.018
RAG (Llama3-70b,k=10)	0.463 ± 0.019	0.474 ± 0.011	0.673 ± 0.052	0.556 ± 0.012	$\sim 10^5$	0.154 ± 0.114
RAG (GPT-oss-20b,k=10)	0.598 ± 0.038	0.563 ± 0.023	**0.880** ± 0.066	0.687 ± 0.035	$\sim 10^4$	0.039 ± 0.004
GraphLogDebugger (Ours)	**0.685** ± 0.065	**0.637** ±0.082	0.870 ± 0.099	**0.734** ± 0.024	40.39	592.073 ± 16.513

Dataset: **HDFS** Task: **Object Anomaly**

Method	Detection Effectiveness				Efficiency	
	Acc.	Prec.	Recall	F1	GFLOPs	Throughput (it/s)
MLP	0.799 ± 0.053	0.801 ± 0.052	0.989 ± 0.000	0.885 ± 0.032	1.4	479.0 ± 194.0
RAG (Llama3-70b,k=5)	0.165 ± 0.022	0 ± 0*	0 ± 0*	0 ± 0*	$\sim 10^5$	0.162 ± 0.085
RAG (GPT-oss-20b,k=5)	0.138 ± 0.029	0 ± 0*	0 ± 0*	0 ± 0*	$\sim 10^4$	0.183 ± 0.010
RAG (Llama3-70b,k=10)	0.173 ± 0.040	0 ± 0*	0 ± 0*	0 ± 0*	$\sim 10^5$	0.194 ± 0.004
RAG (GPT-oss-20b,k=10)	0.138 ± 0.029	0 ± 0*	0 ± 0*	0 ± 0*	$\sim 10^4$	0.192 ± 0.027
GraphLogDebugger (Ours)	**0.989** ± 0.023	**1.000** ± 0.000	**0.987** ± 0.029	**0.993** ± 0.015	5.57	529.999 ± 201.308

Dataset: **Analyst** Task: **Event Anomaly**

Method	Detection Effectiveness				Efficiency	
	Acc.	Prec.	Recall	F1	GFLOPs	Throughput (it/s)
MLP	0.948 ± 0.019	0.922 ± 0.064	0.980 ± 0.043	0.950 ± 0.016	5.58	1996.0 ± 454.0
RAG (Llama3-70b,k=5)	0.408 ± 0.038	0.426 ± 0.026	0.527 ± 0.029	0.471 ± 0.019	$\sim 10^5$	0.204 ± 0.009
RAG (GPT-oss-20b,k=5)	0.770 ± 0.106	0.771 ± 0.157	0.773 ± 0.014	0.772 ± 0.085	$\sim 10^4$	0.145 ± 0.018
RAG (Llama3-70b,k=10)	0.377 ± 0.014	0.400 ± 0.004	0.493 ± 0.038	0.442 ± 0.014	$\sim 10^5$	0.204 ± 0.015
RAG (GPT-oss-20b,k=10)	0.803 ± 0.090	0.798 ± 0.099	0.813 ± 0.087	0.805 ± 0.087	$\sim 10^4$	0.149 ± 0.014
GraphLogDebugger (Ours)	**0.957** ± 0.040	**0.921** ± 0.069	**1.000** ± 0.000	**0.959** ± 0.037	8.97	1037.6 ± 286.2

Dataset: **Landslide** Task: **Object Anomaly**

Method	Detection Effectiveness				Efficiency	
	Acc.	Prec.	Recall	F1	GFLOPs	Throughput (it/s)
MLP	0.831 ± 0.079	0.842 ± 0.180	0.841 ± 0.149	0.838 ± 0.056	5.58	5391.0 ± 328.0
RAG (Llama3-70b,k=5)	0.543 ± 0.074	**0.944** ± 0.056	0.095 ± 0.156	0.168 ± 0.267	$\sim 10^5$	0.355 ± 0.044
RAG (GPT-oss-20b,k=5)	0.611 ± 0.068	0.701 ± 0.050	0.389 ± 0.246	0.495 ± 0.195	$\sim 10^4$	0.155 ± 0.136
RAG (Llama3-70b,k=10)	0.551 ± 0.034	0.904 ± 0.204	0.119 ± 0.102	0.208 ± 0.160	$\sim 10^5$	0.321 ± 0.193
RAG (GPT-oss-20b,k=10)	0.623 ± 0.074	0.723 ± 0.081	0.397 ± 0.149	0.511 ± 0.144	$\sim 10^4$	0.166 ± 0.008
GraphLogDebugger (Ours)	**0.840** ± 0.080	0.798 ± 0.117	**0.929** ± 0.059	**0.858** ± 0.062	19.70	1334.13±108.40

Baselines. Our framework naturally generalizes to tabular log in different domains. Hence, we mainly compare it to baselines which are generally capable of dynamically processing different types of tabular log that contains text-rich and categorical attributes. **MLP** exploits a pretrained text embedding model to embed events and a learnable embedding for objects. A 3-layer MLP is then applied to map the embeddings to anomaly scores. We also compare a series of baselines based on retrieval augmented generation (**RAG**) (Lewis et al., 2020), which is the mainstream method to process general tabular log in the realistic scenario (Akhtar et al., 2025). We deploy RAG based on two advanced open-sourced LLMs, Llama-3-70b (Dubey et al., 2024) and GPT-oss-20b (Agarwal et al., 2025) with the 5 and 10 retrieval entries. The retrieval database is built on the whole training split and the seen log entries during the online evaluation. We construct specified prompts for different datasets and omitted the description to Appendix A. Both MLP and all RAG baselines use all-MiniLM-L6-v2 [3] (Reimers & Gurevych, 2019) as the text embedding model. We do not compare to baselines on log anomaly detection because all these methods are either template-based or domain-specific, which cannot be applied to datasets other than computer system log.

Task. Following the setting of unsupervised anomaly detection (Liu et al., 2021; Schmidl et al., 2022), our basic task is to output an anomaly score for each log entry x_n at timestamp t_n , where higher scores denote more outlyingness (Han et al., 2022). In our task, we use 1 to denote anomalies and 0 to denote normal examples in the ground-truth. We use the first 90% split of the dataset for training, where both the log entries and their anomaly labels are available to access for methods. Methods train the model on this training split or use it for the retrieval database. For the rest 10%, we use it as the test split in our online evaluation, where methods can make use of the seen log entries but their anomaly labels are not accessible. We study two types of anomalies in our experiments: object anomalies and event anomalies. Following the definition in Section 3, we inject object anomalies by swapping an object in the log entry with another existing object. To ensure that historical data contains useful information, we only perturb existing objects in the history. Event anomalies are generated similarly by swapping events. The anomaly rate is set to be 0.05.

Evaluation. We calculate the metrics for information retrieval: accuracy, precision, recall, and f1 score for the dataset. We also evaluate the efficiency of different methods by GFlops and throughputs. During evaluation, we notice that LLM-based baselines tend to be very slow in processing speed. Hence, we include all anomaly log entries and 50 random normal entries in a subset and run RAG only on this subset. For other baselines and our methods, we obtain the prediction result for the full test split but only compute the metric on the above subset for fair comparison. We run experiments three times and post the average value of metrics with error bars.

GraphLogDebugger. The GNN architecture in GraphLogDebugger is a 3-layer GAT backbone with a two-branch node embedder and an MLP prediction head. The node embedder uses 512-d embeddings for objects and the text embedding of all-MiniLM-L6-v2 (Reimers & Gurevych, 2019) for events, with an MLP to map them into the same space. The embedding size of GAT and the prediction head is also 512. We train the GNN for 10 epochs under the learning rate 0.0001 on Adam and the negative ratio 10 on the training split. Following You et al. (2022), we set the batch size as 1 and use a window length of 100 to accelerate the processing.

5.2 Main Results

Table 2 shows that **GraphLogDebugger** consistently outperforms both MLP and RAG-based baselines across all tabular log datasets on five tasks, in terms of detection performance and efficiency.

Effectiveness: GraphLogDebugger achieves the highest F1 scores across all tasks, outperforming RAG baselines—especially in structurally complex domains like HDFS, where RAG methods fail to detect meaningful anomalies (F1 = 0.0). Specifically, RAG baselines are completely fooled by the anomaly pattern that their predicted labels depend on whether there is an existing record with the same format in the retrieved examples, which does not contribute to a reasonable prediction. Two tasks on the Arxiv dataset are the most difficult, where GraphLogDebugger still beats baselines with a higher precision in not abusing anomaly prediction. Even in semantically rich settings such as Analyst and LandSlide, where RAG baselines are expected to excel, our model surpasses them.

Efficiency: RAG approaches exhibit extremely low throughput (typically below 0.3 iterations per second) due to the computational overhead of large language models. In contrast, GraphLogDebugger achieves throughput of at least 500 per second, with significantly lower GFLOPs, enabling real-time anomaly detection in high-throughput environments.

5.3 Case study: Where does RAG fail?

Method	Correlation
RAG	-0.1087
GraphLogDebugger	0.1561

It is natural that GraphLogDebugger yields advantages in efficiency compared to the RAG-baseline, for the latter relies on LLMs with billions of parameters. However, the

[3] https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

leading performance of GraphLogDebugger in detection needs further explanation, while RAG enjoys the general comprehension and reasoning ability of modern LLMs. To this end, we study cases from event anomaly detection of the Arxiv dataset. We choose this task because the degree of event nodes can directly reflect the local graph density of the node-of-interest. We calculate the correlation between event node degrees and the accuracy of GraphLogDebugger and that of RAG(GPT-oss-20b,k=10). The result in the table shows that **the accuracy of GraphLogDebugger is positively correlated with the node degree, while the accuracy of RAG is negatively correlated with the node degree.** This indicates that GraphLogDebugger outperforms RAG on event nodes with rich connections with objects, where semantics of these objects are necessary to detect the anomaly.

Case 1: Label=negative, RAG=positive, GraphLogDebugger=negative
Title: "SymbioSim: Human-in-the-loop Simulation Platform for Bidirectional Continuing Learning in Human-Robot Interaction"
Authors: "Haoran Chen", "Yiming Ren", "Xinran Li", "Ning Ding", "Ziyi Wang", "Yuhan Chen", "Zhiyang Dou", "Yuexin Ma", "Changhe Tu" (9 objects)
Reason (RAG): The author team composition, research domain mismatch, and unclear collaboration patterns raise suspicions about the coherence of the record.
Case 2: Label=positive, RAG=negative, GraphLogDebugger=positive
Title: "VERA: Explainable Video Anomaly Detection via Verbalized Learning of Vision-Language Models"
Authors: "Shubham Gupta", "Zichao Li", "Tianyi Chen", "Cem Subakan", "Siva Reddy", "Perouz Taslakian", "Valentina Zantedeschi" (7 objects)
Reason (RAG): The record seems coherent, with individual authors' expertise areas aligning with the paper's topic, although the team size is slightly larger than expected.
Case 3: Label=positive, RAG=positive, GraphLogDebugger=negative
Title: "Transformer^{-1}: Input-Adaptive Computation for Resource-Constrained Deployment"
Authors: "Yitong Yin" (1 objects)
Reason (RAG): The record consists of a single author, which is consistent with similar papers in the same research domain.

We further raise three cases above to investigate when and how GraphLogDebugger and RAG fail. In Case 1, RAG predicts the normal example as abnormal because the limited retrieved examples do not provide enough evidence to prove the coherence of the author team. By contrast, GraphLogDebugger validates overall team consistency by checking the research background of every author, which correctly predicts the negative label. Case 2 is complementary to Case 1, where GraphLogDebugger is able to scan the research interest of every author and detect the anomaly accurately. However, when the connected objects are few, such as in Case 3, GraphLogDebugger may not have enough references based on the graph to make a correct judgment. In similar cases, RAG could then outperform GraphLogDebugger to recognize patterns in the number of authors in the same domain.

These cases provide insights on how graphs can benefit retrieval augmented generation. When the key entry has dense connections with other entries, traditional retrieval based on similarity cannot efficiently include enough entries to enhance the generation quality. With the help of modern embedding models, graphs can be introduced to gather information in these multi-entry scenarios.

6 Conclusion

We propose a general framework to cover online debugging for heterogeneous tabular logs. By modeling online log debugging as anomaly detection of dynamic graphs, our framework integrates different types of log data into a unified modality by text embedding models, where a dynamic GNN debugs the log through link prediction. Our framework shows good performance in four different datasets while maintaining high efficiency compared to the mainstream RAG-based method.

Limitation. Our work explores combining dynamic GNNs and text embedding models to process log data under the online setting, which indicates the potential to accelerate the online process of data streams by a graph-based method. Nevertheless, our experiments mainly show this potential in the bug detection setting. We leave the exploration of online bug correction to future work.

ETHICS STATEMENT

Our work focuses on detecting inconsistencies in general tabular log data, which enhances the progress of automated log data processing in real-world scenarios. While automation of log processing may raise issues concerning hallucination or fraud reporting, our work does not explicitly introduce new risks compared to existing research.

REPRODUCIBILITY STATEMENT

All implementation details of our method and baselines are given in Section 5 and Appendix A. We will release at the time of publication.

REFERENCES

Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K Arora, Yu Bai, Bowen Baker, Haiming Bao, et al. gpt-oss-120b & gpt-oss-20b model card. *arXiv preprint arXiv:2508.10925*, 2025.

Siraaj Akhtar, Saad Khan, and Simon Parkinson. Llm-based event log analysis techniques: A survey. *arXiv preprint arXiv:2502.00677*, 2025.

Peter W Battaglia, Jessica B Hamrick, Victor Bapst, Alvaro Sanchez-Gonzalez, Vinicius Zambaldi, Mateusz Malinowski, Andrea Tacchetti, David Raposo, Adam Santoro, Ryan Faulkner, et al. Relational inductive biases, deep learning, and graph networks. *arXiv preprint arXiv:1806.01261*, 2018.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901, 2020.

Martina Castillo, Antonio Pecchia, and Ugo Villano. Autolog: Anomaly detection by deep autoencoding of system logs. *Expert Systems with Applications*, 191, 2022.

Jialin Chen and Rex Ying. Tempme: Towards the explainability of temporal graph neural networks via motif discovery. *Advances in Neural Information Processing Systems*, 36:29005–29028, 2023.

Tianyi Chen and Charalampos Tsourakakis. Antibenford subgraphs: Unsupervised anomaly detection in financial networks. In *Proceedings of the 28th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 2762–2770, 2022.

Dawei Cheng, Fangzhou Yang, Xiaoyang Wang, Ying Zhang, and Liqing Zhang. Knowledge graph-based event embedding framework for financial quantitative investments. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 2221–2230, 2020.

Colin B Clement, Matthew Bierbaum, Kevin P O'Keeffe, and Alexander A Alemi. On the use of arxiv as a dataset. *arXiv preprint arXiv:1905.00075*, 2019.

Zihan Dong, Xinyu Fan, and Zhiyuan Peng. Fnspid: A comprehensive financial news dataset in time series, 2024.

Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC conference on computer and communications security*, pp. 1285–1298, 2017.

Xiaoyu Duan, Shi Ying, Wanli Yuan, Hailong Cheng, and Xiang Yin. A generative adversarial networks for log anomaly detection. *Computer Systems Science & Engineering*, 37(1), 2021.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. The llama 3 herd of models. *arXiv e-prints*, pp. arXiv–2407, 2024.

Vijay Prakash Dwivedi, Chaitanya K Joshi, Thomas Laurent, Yoshua Bengio, and Xavier Bresson. Benchmarking graph neural networks. *arXiv preprint arXiv:2003.00982*, 2020.

Ocheme Anthony Ekle and William Eberle. Anomaly detection in dynamic graphs: A comprehensive survey. *ACM Transactions on Knowledge Discovery from Data*, 18(8):1–44, 2024.

Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. Large language models (llms) on tabular data: Prediction, generation, and understanding–a survey. *arXiv preprint arXiv:2402.17944*, 2024.

Tao Feng, Yexin Wu, Guanyu Lin, and Jiaxuan You. Graph world model. *arXiv preprint arXiv:2507.10539*, 2025.

Josh Gardner, Juan C Perdomo, and Ludwig Schmidt. Large scale transfer learning for tabular data via language modeling. *Advances in Neural Information Processing Systems*, 37:45155–45205, 2024.

Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pp. 1263–1272. Pmlr, 2017.

Haixuan Guo, Shuhan Yuan, and Xintao Wu. Logbert: Log anomaly detection via bert. In *2021 international joint conference on neural networks (IJCNN)*, pp. 1–8. IEEE, 2021.

Will Hamilton, Zhitao Ying, and Jure Leskovec. Inductive representation learning on large graphs. *Advances in neural information processing systems*, 30, 2017.

Songqiao Han, Xiyang Hu, Hailiang Huang, Minqi Jiang, and Yue Zhao. Adbench: Anomaly detection benchmark. *Advances in neural information processing systems*, 35:32142–32159, 2022.

Shilin He, Jieming Zhu, Pinjia He, and Michael R Lyu. Experience report: System log analysis for anomaly detection. In *2016 IEEE 27th international symposium on software reliability engineering (ISSRE)*, pp. 207–218. IEEE, 2016.

Zhangyue He, Yanni Tang, Kaiqi Zhao, Jiamou Liu, and Wu Chen. Graph-based log anomaly detection via adversarial training. In *International Symposium on Dependable Software Engineering: Theories, Tools, and Applications*, pp. 55–71. Springer, 2023.

Aidan Hogan, Eva Blomqvist, Michael Cochez, Claudia d'Amato, Gerard De Melo, Claudio Gutierrez, Sabrina Kirrane, José Emilio Labra Gayo, Roberto Navigli, Sebastian Neumaier, et al. Knowledge graphs. *ACM Computing Surveys (Csur)*, 54(4):1–37, 2021.

Shenyang Huang, Farimah Poursafaei, Jacob Danovitch, Matthias Fey, Weihua Hu, Emanuele Rossi, Jure Leskovec, Michael Bronstein, Guillaume Rabusseau, and Reihaneh Rabbany. Temporal graph benchmark for machine learning on temporal graphs. *Advances in Neural Information Processing Systems*, 36:2056–2073, 2023.

Alistair EW Johnson, Lucas Bulgarelli, Lu Shen, Alvin Gayles, Ayad Shammout, Steven Horng, Tom J Pollard, Sicheng Hao, Benjamin Moody, Brian Gow, et al. Mimic-iv, a freely accessible electronic health record dataset. *Scientific data*, 10(1):1, 2023.

Rodney Kinney, Chloe Anastasiades, Russell Authur, Iz Beltagy, Jonathan Bragg, Alexandra Buraczynski, Isabel Cachola, Stefan Candra, Yoganand Chandrasekhar, Arman Cohan, et al. The semantic scholar open data platform. *arXiv preprint arXiv:2301.10140*, 2023.

TN Kipf. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*, 2016.

Ross Koval, Nicholas Andrews, and Xifeng Yan. Financial forecasting from textual and tabular time series. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 8289–8300, 2024.

Srijan Kumar, Xikun Zhang, and Jure Leskovec. Predicting dynamic embedding trajectory in temporal interaction networks. In *Proceedings of the 25th ACM SIGKDD international conference on knowledge discovery & data mining*, pp. 1269–1278, 2019.

Litton Jose Kurisinkel, Pruthwik Mishra, and Yue Zhang. Text2timeseries: Enhancing financial forecasting through time series prediction updates with event-driven insights from large language models. *arXiv preprint arXiv:2407.03689*, 2024.

Yukyung Lee, Jina Kim, and Pilsung Kang. Lanobert: System log anomaly detection based on bert masked language model. *Applied Soft Computing*, 146:110689, 2023.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Guohao Li, Matthias Müller, Ali Thabet, and Bernard Ghanem. Deepgcns: Can gcns go as deep as cnns? *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2021.

Yaguang Li, Rose Yu, Cyrus Shahabi, and Yan Liu. Diffusion convolutional recurrent neural network: Data-driven traffic forecasting. *arXiv preprint arXiv:1707.01926*, 2017.

Yufei Li, Yanchi Liu, Haoyu Wang, Zhengzhang Chen, Wei Cheng, Yuncong Chen, Wenchao Yu, Haifeng Chen, and Cong Liu. Glad: Content-aware dynamic graphs for log anomaly detection. In *2023 IEEE International Conference on Knowledge Graph (ICKG)*, pp. 9–18. IEEE, 2023.

Tianyang Liu, Fei Wang, and Muhao Chen. Rethinking tabular data understanding with large language models. *arXiv preprint arXiv:2312.16702*, 2023.

Yilun Liu, Shimin Tao, Weibin Meng, Feiyu Yao, Xiaofeng Zhao, and Hao Yang. Logprompt: Prompt engineering towards zero-shot and interpretable log analysis. In *Proceedings of the 2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings*, pp. 364–365, 2024.

Yixin Liu, Zhao Li, Shirui Pan, Chen Gong, Chuan Zhou, and George Karypis. Anomaly detection on attributed networks via contrastive self-supervised learning. *IEEE transactions on neural networks and learning systems*, 33(6):2378–2392, 2021.

Alessandro Longa, Valerio Lachi, Giovanni Santin, Monica Bianchini, Bruno Lepri, et al. Graph neural networks for temporal graphs: State of the art, open challenges, and opportunities. *arXiv preprint arXiv:2305.12472*, 2023.

Siyang Lu, Xiang Wei, Yandong Li, and Liqiang Wang. Detecting anomaly in big data system logs using convolutional neural network. In *2018 IEEE 16th Intl Conf on Dependable, Autonomic and Secure Computing, 16th Intl Conf on Pervasive Intelligence and Computing, 4th Intl Conf on Big Data Intelligence and Computing and Cyber Science and Technology Congress (DASC/PiCom/-DataCom/CyberSciTech)*, pp. 151–158. IEEE, 2018.

Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, et al. Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs. In *IJCAI*, volume 19, pp. 4739–4745, 2019.

Geon No et al. Training-free retrieval-based log anomaly detection with pre-trained language models. *Engineering Applications of Artificial Intelligence*, 133, 2024.

Jonathan Pan, Wong Swee Liang, and Yuan Yidi. Raglog: Log anomaly detection using retrieval augmented generation. In *2024 IEEE World Forum on Public Safety Technology (WFPST)*, pp. 169–174. IEEE, 2024.

Guansong Pang, Chunhua Shen, Longbing Cao, and Anton Van Den Hengel. Deep learning for anomaly detection: A review. *ACM computing surveys (CSUR)*, 54(2):1–38, 2021.

Yunhe Pang, Bo Chen, Fanjin Zhang, Yanghui Rao, Evgeny Kharlamov, and Jie Tang. Guard: Effective anomaly detection through a text-rich and graph-informed language model. In *Proceedings of the 31st ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 2*, pp. 2222–2233, 2025.

Aldo Pareja, Giacomo Domeniconi, Jie Chen, Tengfei Ma, Toyotaro Suzumura, Hiroki Kanezashi, Tim Kaler, Tao Schardl, and Charles Leiserson. Evolvegcn: Evolving graph convolutional networks for dynamic graphs. In *Proceedings of the AAAI conference on artificial intelligence*, volume 34, pp. 5363–5370, 2020.

Taejin Park. Enhancing anomaly detection in financial markets with an llm-based multi-agent framework. *arXiv preprint arXiv:2403.19735*, 2024.

Yulong Pei, Fang Lyu, Werner Van Ipenburg, and Mykola Pechenizkiy. Subgraph anomaly detection in financial transaction networks. In *Proceedings of the First ACM International Conference on AI in Finance*, pp. 1–8, 2020.

Jiaxing Qi, Shaohan Huang, Zhongzhi Luan, Shu Yang, Carol Fung, Hailong Yang, Depei Qian, Jing Shang, Zhiwen Xiao, and Zhihui Wu. Loggpt: Exploring chatgpt for log-based anomaly detection. In *2023 IEEE International Conference on High Performance Computing & Communications, Data Science & Systems, Smart City & Dependability in Sensor, Cloud & Big Data Systems & Application (HPCC/DSS/SmartCity/DependSys)*, pp. 273–280. IEEE, 2023.

Nils Reimers and Iryna Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019. URL https://arxiv.org/abs/1908.10084.

Emanuele Rossi, Ben Chamberlain, Fabrizio Frasca, Davide Eynard, Federico Monti, and Michael Bronstein. Temporal graph networks for deep learning on dynamic graphs. *arXiv preprint arXiv:2006.10637*, 2020.

Eduardo J Ruiz, Vagelis Hristidis, Carlos Castillo, Aristides Gionis, and Alejandro Jaimes. Correlating financial time series with micro-blogging activity. In *Proceedings of the fifth ACM international conference on Web search and data mining*, pp. 513–522, 2012.

Aravind Sankar, Yanhong Wu, Liang Gou, Wei Zhang, and Hao Yang. Dysat: Deep neural representation learning on dynamic graphs via self-attention networks. In *Proceedings of the 13th international conference on web search and data mining*, pp. 519–527, 2020.

Sebastian Schmidl, Phillip Wenig, and Thorsten Papenbrock. Anomaly detection in time series: a comprehensive evaluation. *Proceedings of the VLDB Endowment*, 15(9):1779–1797, 2022.

Youngjoo Seo, Michaël Defferrard, Pierre Vandergheynst, and Xavier Bresson. Structured sequence modeling with graph convolutional recurrent networks. In *International conference on neural information processing*, pp. 362–373. Springer, 2018.

Xiaoming Shi, Siqiao Xue, Kangrui Wang, Fan Zhou, James Zhang, Jun Zhou, Chenhao Tan, and Hongyuan Mei. Language models can improve event prediction by few-shot abductive reasoning. *Advances in Neural Information Processing Systems*, 36:29532–29557, 2023.

Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, Zhengdan Li, Yongqian Sun, Fangrui Guo, Junyu Shen, Yuzhi Zhang, et al. Logkg: Log failure diagnosis through knowledge graph. *IEEE Transactions on Services Computing*, 16(5):3493–3507, 2023.

Paul C Tetlock. Giving content to investor sentiment: The role of media in the stock market. *The Journal of finance*, 62(3):1139–1168, 2007.

Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. *arXiv preprint arXiv:1710.10903*, 2017.

Jingru Wang, Wen Ding, and Xiaotong Zhu. Financial analysis: Intelligent financial data analysis system based on llm-rag. *arXiv preprint arXiv:2504.06279*, 2025.

Xinlei Wang, Maike Feng, Jing Qiu, Jinjin Gu, and Junhua Zhao. From news to forecast: Integrating event analysis in llm-based time series forecasting with reflection. *Advances in Neural Information Processing Systems*, 37:58118–58153, 2024a.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, et al. Chain-of-table: Evolving tables in the reasoning chain for table understanding. *arXiv preprint arXiv:2401.04398*, 2024b.

Xiaofeng Wu, Alan Ritter, and Wei Xu. Tabular data understanding with llms: A survey of recent advances and challenges. *arXiv preprint arXiv:2508.00217*, 2025.

Mengxi Xiao, Zihao Jiang, Lingfei Qian, Zhengyu Chen, Yueru He, Yijing Xu, Yuecheng Jiang, Dong Li, Ruey-Ling Weng, Min Peng, et al. Enhancing financial time-series forecasting with retrieval-augmented large language models. *arXiv preprint arXiv:2503.67890*, 2025.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. Inductive representation learning on temporal graphs. *arXiv preprint arXiv:2002.07962*, 2020.

Wei Xu, Ling Huang, Armando Fox, David Patterson, and Michael I Jordan. Detecting large-scale system problems by mining console logs. In *Proceedings of the ACM SIGOPS 22nd symposium on Operating systems principles*, pp. 117–132, 2009.

Hang Yang, Yubo Chen, Kang Liu, Yang Xiao, and Jun Zhao. Dcfee: A document-level chinese financial event extraction system based on automatically labeled training data. In *Proceedings of ACL 2018, System Demonstrations*, pp. 50–55, 2018.

Lin Yang, Junjie Chen, Zan Wang, Weijing Wang, Jiajun Jiang, Xuyuan Dong, and Wenbin Zhang. Semi-supervised log-based anomaly detection via probabilistic label estimation. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*, pp. 1448–1460. IEEE, 2021.

Chenchen Ye, Ziniu Hu, Yihe Deng, Zijie Huang, Mingyu Derek Ma, Yanqiao Zhu, and Wei Wang. Mirai: Evaluating llm agents for event forecasting. *arXiv preprint arXiv:2407.01231*, 2024.

Jiaxuan You, Tianyu Du, and Jure Leskovec. Roland: graph learning framework for dynamic graphs. In *Proceedings of the 28th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 2358–2366, 2022.

Xinli Yu, Zheng Chen, Yuan Ling, Shujing Dong, Zongyi Liu, and Yanbin Lu. Temporal data meets llm–explainable financial time series forecasting. *arXiv preprint arXiv:2306.11025*, 2023.

Samreen Zehra, Syed Farhan Mohsin Mohsin, Shaukat Wasi, Syed Imran Jami, Muhammad Shoaib Siddiqui, and Muhammad Khaliq-Ur-Rahman Raazi Syed. Financial knowledge graph based financial report query system. *IEEE Access*, 9:69766–69782, 2021.

Jiasheng Zhang, Jialin Chen, Menglin Yang, Aosong Feng, Shuang Liang, Jie Shao, and Rex Ying. Dtgb: A comprehensive benchmark for dynamic text-attributed graphs. *Advances in Neural Information Processing Systems*, 37:91405–91429, 2024a.

Lingzhe Zhang, Tong Jia, Mengxi Jia, Yifan Wu, Hongyi Liu, and Ying Li. Xraglog: A resource-efficient and context-aware log-based anomaly detection method using retrieval-augmented generation. In *AAAI 2025 Workshop on Preventing and Detecting LLM Misinformation (PDLM)*, 2025a.

Linming Zhang, Wenzhong Li, Zhijie Zhang, Qingning Lu, Ce Hou, Peng Hu, Tong Gui, and Sanglu Lu. Logattn: Unsupervised log anomaly detection with an autoencoder based attention mechanism. In *International conference on knowledge science, engineering and management*, pp. 222–235. Springer, 2021.

Shaokun Zhang, Ming Yin, Jieyu Zhang, Jiale Liu, Zhiguang Han, Jingyang Zhang, Beibin Li, Chi Wang, Huazheng Wang, Yiran Chen, et al. Which agent causes task failures and when? on automated failure attribution of llm multi-agent systems. *arXiv preprint arXiv:2505.00212*, 2025b.

Xiaokang Zhang, Sijia Luo, Bohan Zhang, Zeyao Ma, Jing Zhang, Yang Li, Guanlin Li, Zijun Yao, Kangli Xu, Jinchang Zhou, et al. Tablellm: Enabling tabular data manipulation by llms in real office usage scenarios. *arXiv preprint arXiv:2403.19318*, 2024b.

Xinye Zhang, Xiaoli Chai, Minghua Yu, and Ding Qiu. Anomaly detection model for log based on lstm network and variational autoencoder. In *2023 4th International Conference on Information Science, Parallel and Distributed Systems (ISPDS)*, pp. 239–244. IEEE, 2023a.

Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM joint meeting on European software engineering conference and symposium on the foundations of software engineering*, pp. 807–817, 2019.

Zeyang Zhang, Xin Wang, Ziwei Zhang, Zhou Qin, Weigao Wen, Hui Xue, Haoyang Li, and Wenwu Zhu. Spectral invariant learning for dynamic graphs under distribution shifts. *Advances in Neural Information Processing Systems*, 36:6619–6633, 2023b.

Qihang Zhao and Xiaodong Feng. Utilizing citation network structure to predict paper citation counts: A deep learning approach. *Journal of Informetrics*, 16(1):101235, 2022.

Aoxiao Zhong, Dengyao Mo, Guiyang Liu, Jinbu Liu, Qingda Lu, Qi Zhou, Jiesheng Wu, Quanzheng Li, and Qingsong Wen. Logparser-llm: Advancing efficient log parsing with large language models. In *Proceedings of the 30th ACM SIGKDD Conference on Knowledge Discovery and Data Mining*, pp. 4559–4570, 2024.

Jieming Zhu, Shilin He, Pinjia He, Jinyang Liu, and Michael R Lyu. Loghub: A large collection of system log datasets for ai-driven log analytics. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*, pp. 355–366. IEEE, 2023a.

Yifan Zhu, Fangpeng Cong, Dan Zhang, Wenwen Gong, Qika Lin, Wenzheng Feng, Yuxiao Dong, and Jie Tang. Wingnn: Dynamic graph neural networks with random gradient aggregation window. In *Proceedings of the 29th ACM SIGKDD conference on knowledge discovery and data mining*, pp. 3650–3662, 2023b.

A Appendix

A.1 Use of LLMs

We use ChatGPT to polish our introduction (Section 1) and generate the notation table (Table 3), both of which have been checked manually. We also use ChatGPT to retrieve related works in the tabular log processing part by searching machine-learning based log processing methods.

A.2 Notation

Table 3: **Notation**

Symbol	**Type**	**Meaning**
Tabular-log basics (Sec. 3)		
$X = \{x_0, \ldots, x_{N-1}\}$	sequence	Time-ordered tabular log (entries).
x_n	entry	The n-th log entry.
$t_0 < \cdots < t_{N-1}$	timestamps	Arrival times of entries.
x_n^m	attribute value	The m-th attribute in entry x_n.
M	integer	Number of attributes per entry.
$\{o_n^0, \ldots, o_n^{P-1}\}$	set	Object attributes extracted from x_n.
P	integer	Number of object attributes in x_n ($P < M$).
s_n	text / node	Event attribute (one per entry; possibly text).
$\{f_n^0, \ldots, f_n^{Q-1}\}$	set	Feature attributes extracted from x_n.

Continued on next page

Symbol	Type	Meaning
Q	integer	Number of feature attributes in x_n ($Q < M$).
y_n	label	Event anomaly label for x_n (0 normal, 1 abnormal).
y_n^p	label	Object anomaly label for object o_n^p $(0/1)$.
y_n^q	label	Feature anomaly label for feature f_n^q $(0/1)$.
Graphs and dynamics (Sec. 4.1–4.2)		
$\mathcal{G}$	dynamic graph	Evolving heterogeneous graph over time.
G_n	snapshot	Graph snapshot at time t_n (before merging g_n).
g_n	subgraph	Subgraph constructed from new entry x_n.
$G_{n+1} \setminus G_n$	graph diff	Increment between consecutive snapshots; here equal to g_n.
$\mathcal{V}, \mathcal{E}$	sets	Node and edge sets of the current graph.
v, e	node, edge	A node or an edge (generic).
$\mathcal{V}_n^o$	node set	Object nodes appearing in x_n.
$\mathcal{V}_n^e$	node set	New event nodes introduced by x_n (events are unique).
$\mathcal{E}_k^+$	edge set	Positive (observed) object–event links in g_k.
$\mathcal{E}_k^-$	edge set	Negative samples (non-existent object–event pairs).
$\hat{\mathcal{E}}_n^+$	edge set	Accepted/predicted-positive links at t_n.
$\hat{\mathcal{V}}_n^e$	node set	Accepted new events incident to $\hat{\mathcal{E}}_n^+$.
$\mathcal{R}_{t_n}$	set	Per-time link predictions/results at t_n.
$\{G_n\}_{n=0}^{N-1}$	sequence	The sequence of snapshots defining $\mathcal{G}$.
G_{t_K}, G_{t_N}	snapshots	Snapshot after train time t_K, and final snapshot at t_N.
Modeling (GNN and scoring; Sec. 4.3)		
θ	parameters	Trainable parameters of the GNN.
$f_\theta(\cdot)$	mapping	GNN that computes object-node embeddings on $\mathcal{G}$.
$\mathcal{F}$	encoder	Text (and time-aware) embedding model for events.
$h_o,\ h_e$	vectors	Object and event embeddings, respectively.
$\mathrm{reduce}(\cdot,\cdot)$	operator	Embedding combiner (e.g., concat/diff/dot).
$\mathrm{MLP}(\cdot)$	mapping	Multi-layer perceptron used for scoring.
$\sigma(\cdot)$	function	Sigmoid activation.
$s_{o,e}$	score	Link-normality score for pair (o, e).
$\hat{\ell}_{o,e}$	label	Predicted link label: $\mathbb{1}[s_{o,e} \geq \tau]$.
$\mathcal{L}_k$	loss	Balanced BCE loss at training step k.
η	scalar	Learning rate.
τ	threshold	Operating threshold for prediction.
ρ	ratio	Negative sampling ratio.
Data splits and indices		
$X_{\text{train}},\ X_{\text{test}}$	sequences	Training and test splits (chronological).
K	integer	Index/time that separates train and test.
N	integer	Total number of entries/snapshots.
k, n	indices	Training step k, evaluation time n.
$t_k,\ t_n$	timestamps	Times associated with steps/entries.

A.3 MODEL DESIGN SPACE

We compare two variants in our experiments: (i) Plain (ungated) GAT. We first concatenate the entity-type and entity-ID embeddings and pass them through a feed-forward projection to obtain the initial representation e_0. We then run multi-layer, multi-head GATConv on an entity–entity graph induced by shared content to propagate messages and obtain e_{GAT}, which we use as the final entity representation. (ii) Gated fusion. Starting from the same e_0 and e_{GAT}, we introduce a global learnable scalar gate α and adaptively combine them via a sigmoid: $e = (1-\sigma(\alpha)), e_0 + \sigma(\alpha), e_{\text{GAT}}$. This biases toward e_0 when the given signal is weak (or absent) and toward e_{GAT} when the signal is strong. Both variants share the same link-prediction head: we take the entity representation and the content representation (text and time embeddings concatenated and then projected), compute their element-wise difference, and feed it to an MLP to output the link probability

A.4 Additional Visualization

Figure 3 visualizes the distribution of anomaly likelihood scores of our five evaluation tasks. The score distribution corroborates the main result in Table 2, that Analyst, Arxiv (Node), and HDFS are three tasks relatively easy, with the score distribution of anomalies and normal examples separate clearly. By contrast, the score of anomalies and normal examples mix up in Arxiv (Edge) and Landslide, indicating that these datasets are more difficult.

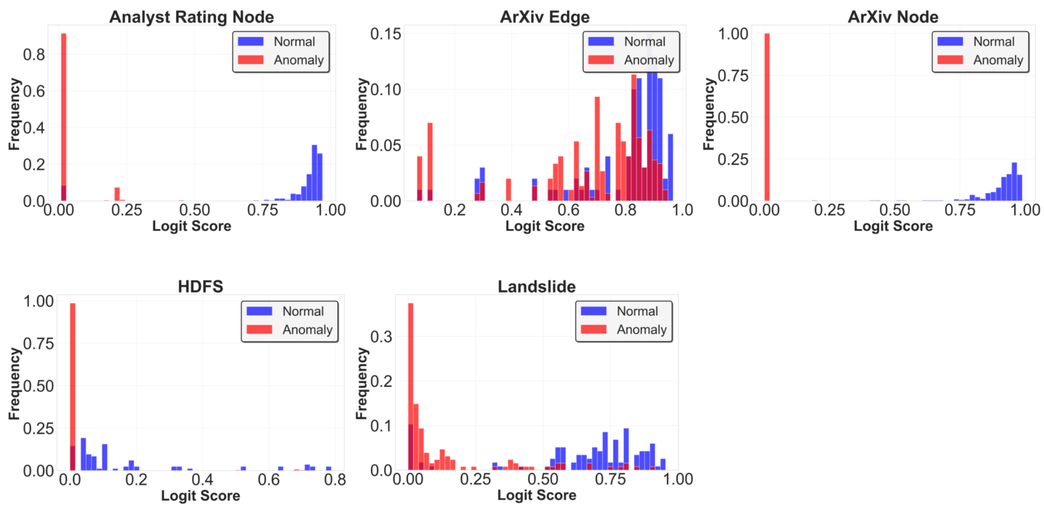

Figure 3: **Anomaly score distribution of five tasks by GraphLogDebugger.** Score distributions of anomalies and normal examples separate for simpler tasks and mix up for more difficult tasks.

A.5 Prompts in RAG

We list the prompt we used in our RAG baseline as follows:

```
"""Build context for ArXiv dataset (authors and paper titles)."""
context = """You are an expert at analyzing author-paper relationships in
    academic research.

DATASET CONTEXT: This is a dataset of academic papers with their authors
    and titles.
- Entities (authors): Research authors who wrote the papers
- Content (titles): The titles of the academic papers
- Edge: A connection between an author and a paper title (indicating the
    author contributed to that paper)

TASK: Determine if the specific author-paper connection (edge) should
    exist based on historical patterns.

EDGE ANALYSIS TARGET:
"""

context += f"Author: {entity_name}\n"
context += f"Paper Title: {content_name}\n\n"

if similar_contents:
    context += "SIMILAR PAPERS AND THEIR AUTHORS (for reference):\n"
    context += "Use these examples to understand what types of authors
        typically work on similar papers.\n\n"

```

```
    for i, content_record in enumerate(similar_contents[:10]):
        content = content_record.get('content', '')
        entities = content_record.get('related_entities', [])
        similarity = content_record.get('similarity', 0.0)
        num_records = content_record.get('num_records', 0)

        context += f"{i+1}. Paper Title: {content} (Similarity: {similarity:.3f}, {num_records} records)\n"
        context += f"   Authors who worked on this paper: {', '.join(entities) if entities else 'None'}\n\n"
else:
    context += "No similar papers found in historical data.\n\n"

context += """ANALYSIS QUESTION:
Based on the similar papers and their author patterns, should the specified author-paper connection exist?

EVALUATION CRITERIA:
1. Research Domain Match: Does the author's expertise align with the paper's topic?
2. Historical Patterns: Do authors with similar expertise appear in similar papers?
3. Authorship Likelihood: Is it reasonable that this author would contribute to this type of research?
4. Anomaly Detection: Does this connection seem unusual or out of place compared to patterns in similar papers?

DECISION GUIDELINES:
- edge_exists = True: The author-paper connection makes sense based on research area and historical patterns
- edge_exists = False: The author seems misplaced or unlikely to work on this type of paper (anomalous edge)
- Consider the research fields, methodologies, and typical author patterns shown in similar papers
- An edge is anomalous if the author appears completely unrelated to the research domain of the paper

CONFIDENCE SCORING:
- High confidence (0.8-1.0): Clear patterns in similar papers strongly support/reject the connection
- Medium confidence (0.5-0.7): Some evidence but less certain
- Low confidence (0.0-0.4): Limited historical data or unclear patterns
"""
```

Listing 1: Prompt: Arxiv

```
"""Build context for HDFS dataset (BlockId focus for detection)."""
context = """You are an expert at analyzing BlockId-log relationships in HDFS distributed file system logs.

DATASET CONTEXT: This is a dataset of HDFS system logs with their Block IDs and log contents.
- Primary Focus: Block IDs (e.g., blk_8215417782549978040, blk_161475555609545016) - unique identifiers for HDFS data blocks
- Content (logs): The actual log messages and operations in the HDFS system that involve specific blocks
- Edge: A connection between a Block ID and a log message (indicating the block is involved in that log operation)

SPECIAL NOTE: For HDFS anomaly detection, we focus specifically on Block ID connections to log messages.
Block IDs should appear BOTH in the BlockId column AND within the log content itself.

```

```
TASK: Determine if the specific Block ID-log connection (edge) should exist based on historical patterns.

EDGE ANALYSIS TARGET:
"""

context += f"Block ID: {entity_name}\n"
context += f"Log Content: {content_name}\n"
context += f"Content Analysis: Does '{entity_name}' appear in the log content? {'YES' if entity_name in content_name else 'NO'}\n\n"

if similar_contents:
    context += "SIMILAR LOG MESSAGES AND THEIR BLOCK IDs (for reference):\n"
    context += "Use these examples to understand what types of Block IDs typically appear in similar log messages.\n\n"

    for i, content_record in enumerate(similar_contents[:10]):
        content = content_record.get('content', '')
        entities = content_record.get('related_entities', [])
        similarity = content_record.get('similarity', 0.0)
        num_records = content_record.get('num_records', 0)

        block_ids = [e for e in entities if e.startswith('blk_')]
        other_entities = [e for e in entities if not e.startswith('blk_')]

        context += f"{i+1}. Log Content: {content} (Similarity: {similarity:.3f}, {num_records} records)\n"
        context += f"   Block IDs in this log: {', '.join(block_ids) if block_ids else 'None'}\n"
        if other_entities:
            context += f"   Other entities: {', '.join(other_entities[:3])}{'...' if len(other_entities) > 3 else ''}\n"
        context += "\n"
else:
    context += "No similar log messages found in historical data.\n\n"

context += """ANALYSIS QUESTION:
Based on the similar log messages and their Block ID patterns, should the specified Block ID-log connection exist?

EVALUATION CRITERIA:
1. Block ID Presence: Does the Block ID appear within the log content itself? (This is crucial for HDFS)
2. Log Operation Match: Does the Block ID relate to the HDFS operation described in the log?
3. Historical Patterns: Do similar Block IDs appear in similar log messages?
4. HDFS Block Behavior: Is it reasonable that this Block ID would be involved in this type of operation?
5. Content Consistency: Block ID should be consistent between the BlockId column and the log content

DECISION GUIDELINES:
- edge_exists = True: The Block ID-log connection makes sense based on HDFS block operations and historical patterns
- edge_exists = False: The Block ID seems unrelated to this log message (anomalous edge)
- CRITICAL: If the Block ID does NOT appear in the log content, this is likely anomalous
- Consider HDFS block operations like allocation, storage, replication shown in similar messages
- An edge is anomalous if the Block ID appears completely unrelated to the log operation
```

```
CONFIDENCE SCORING:
- High confidence (0.8-1.0): Clear Block ID patterns and content
    consistency strongly support/reject the connection
- Medium confidence (0.5-0.7): Some evidence but less certain about Block
     ID relevance
- Low confidence (0.0-0.4): Limited historical data or unclear Block ID
    patterns

IMPORTANT: Focus specifically on Block ID relationships - Components and
    Event IDs are secondary for this analysis.
"""
```

Listing 2: Prompt: HDFS

```
"""Build generic context for unknown datasets."""
context = f"""You are an expert at analyzing entity-content relationships
    .

EDGE ANALYSIS TARGET:
Entity: {entity_name}
Content: {content_name}

TASK: Determine if this entity-content connection should exist based on
    historical patterns.
"""

if similar_contents:
    context += "\nSIMILAR EXAMPLES:\n"
    for i, content_record in enumerate(similar_contents[:5]):
        content = content_record.get('content', '')
        entities = content_record.get('related_entities', [])
        context += f"{i+1}. Content: {content}\n   Related entities: {',
            '.join(entities)}\n\n"

context += """
DECISION: Should this entity-content connection exist?
- edge_exists = True: The connection makes sense based on patterns
- edge_exists = False: The connection seems anomalous
"""
```

Listing 3: Prompt: Analyst and Landslide