Computing and Informatics, Vol. 45, 2026, 203–230, doi: 10.31577/cai_2026_1_203

MULTI-PERSPECTIVE APPROACH FOR ANOMALOUS BEHAVIOR DETECTION AND REPAIRING

Lu Li

School of Mathematics and Physics
Anhui Jianzhu University
Hefei Anhui 230601, China
e-mail: lilu@ahjzu.edu.cn

Huan Fang*, Shouzheng Zhang

School of Mathematics and Big Data
Anhui University of Science and Technology
Huainan Anhui 232001, China
e-mail: {2003122, 2024201449}@aust.edu.cn

Abstract. Event logs of business systems are often dirty owing to recording errors or system conventions, and this dirty data can definitely affect the quality of the event logs. Consequently, many anomalous behaviors that deviate from preestablished process models are frequently seen in event logs. To properly assess and enhance business processes, it is imperative to identify and correct such anomalous behaviors. When the missing or anomalous location is known, the current repair methods have a high success rate; however, when the missing or anomalous location is unknown, they do not function well. Therefore, a formalized solution using an activity feature graph is proposed for event logs where uncertainty arises from missing, duplicate, and replaced occurrences. Several prediction and repair models are used in conjunction with the viewpoints of control flow and data flow to identify and address anomalous behaviors in event logs. In addition to fixing exceptions at the case level, the proposed repairing approach can also correct exceptions at the attribute level, or feature repairing. Process analysts are provided with an activity feature graph to examine and refine the repair outcomes. Finally, four event logs with various anomaly ratios are used to confirm the method's viability.

* Corresponding author

Keywords: Multi-perspective log repairing, activity feature graph, anomaly detection, behavioral repair, feature repair

Mathematics Subject Classification 2010: 68-T20

1 INTRODUCTION

Process mining technology has been widely applied in modern information systems, such as manufacturing, logistics, finance, and healthcare [1]. Process mining technology can help business organizations understand their actual workflow, improve processes, achieve automation, and ultimately enhance their business capabilities and competitiveness [2]. This technology primarily involves analyzing event log data to automatically discover business processes within the organization and generate corresponding process models. It helps managers to identify issues and bottlenecks within the process, and then optimize and improve the process accordingly. Event logs are generated by information systems, and several event occurrences constitute a trace, which represents an ordered sequence of events. Generally, each event includes activities, timestamps, and other attributes to characterize the feature of the event.

Indeed, there have been several process model discovery algorithms developed, such as the alpha miner [3], fuzzy miner [4], heuristic miner [5], and inductive miner [6]. These algorithms can ensure the generation of accurate process models under specific circumstances. However, current process mining algorithms are mainly designed to generate a holistic process model based on the entire event log data. They lack the ability to detect and repair individual behavior within the log. Additionally, these process discovery algorithms mainly focus on control flow, without considering the involvement of data flow.

The existing technologies have addressed the issue of missing event attributes in event logs through various techniques, such as structure-based imputation, clustering, and deep learning algorithms. All of them are designed to ensure the integrity and accuracy of the logs. Furthermore, deep learning methods have been effective in repairing abnormal attributes that may exist in the logs, ensuring the usability of log analysis. However, current techniques lack the ability to detect and repair individual random abnormal behaviors. For instance, issues such as event loss, event duplication, and inconsistent event arrival order during network transmissions can result in abnormal behaviors. These issues cannot be fully repaired solely through attribute analysis and may affect the integrity of the behavior.

The main contributions of this paper include the following aspects:

1. The proposal of a formalized structure called the activity feature graph of event log, which integrates the control flow and data flow perspectives of an event log. This structure can be input into different learning models to achieve applications in different scenarios.

2. The development of a detection and repair method for abnormal behaviors based on the activity feature graph. Additionally, a comparison of the differences between abnormal and normal behaviors is provided.

The remaining sections of this paper are organized as follows: Section 2 describes the background and related work, Section 3 introduces some preliminary knowledge used in this paper, Section 4 presents the proposed method in detail, Section 5 verifies the method through case studies based on various event logs. Finally, in Section 6, we summarize and discuss our research findings.

2 RELATED WORK

In the field of process mining, despite the continuous development of process mining techniques, if the quality of the event logs used for analysis is poor, the quality of the analysis results will still be low even when using good process mining techniques. The problem of event log quality was first raised in [7]. In order to improve the quality of process mining analysis, the work in [8] and [9] developed a process mining methodology as a guide for projects using event logs for process mining, with a focus on data cleaning steps. The work of [10] categorized event log quality problems into four types: missing data, incorrect data, imprecise data, and irrelevant data. In this paper, we mainly categorize the repair of existing logs into two types: attribute-level repair and case-level repair.

2.1 Attribute-Level Repair

The repair at the attribute level involves explicitly identifying the missing positions of attributes, including activity attributes or determining the positions of abnormal attributes based on anomaly detection of attributes. Specifically, as shown in problem 1 (denoted as ①) in Figure 1, the entire table represents a case in the log, where each row represents an event, and each cell represents the attributes contained in the event. The red cells in Figure 1 indicate detected abnormal attributes, while the gray cells indicate missing attributes. The work of [11] focuses on detecting abnormal values and reconstructing missing values at the attribute level in event logs. It consists of two processes: log cleaning and reconstruction. In event log cleaning, abnormal values are those with high reconstruction errors in the autoencoder decoding step. In event log reconstruction, the autoencoder is used to reconstruct missing values in the input dataset. This method does not rely on any prior knowledge of the business process that generates the event log and has shown significant performance in terms of activity labels and timestamps in artificial event logs. In PROELR [12] and SRBA [13], trace clustering methods are used to cluster complete traces, which are traces without any missing activity labels. Each incomplete trace, referring to traces with missing activity labels, is assigned to the nearest cluster. Subsequently, the incomplete traces are repaired based on the characteristics of the

corresponding trace clusters. It is worth noting that both [12] and [13] can only repair missing values at the individual activity level. MIEC [14] is a likelihood-based multiple imputation technique for repairing missing data in event logs. In addition to repairing missing activity labels, it can also repair all other missing attributes in the event log. MIEC relies on the dependencies between event attributes. For example, certain activities may always occur on weekends or be performed by specific groups of people. When such dependencies do not exist or the event log contains limited attribute data, effective repair of the event log may not be possible. [15] applies a decision tree learning algorithm to discover rules for missing values in event logs. [16] proposes a novel classification event imputation method that can recover missing categorical events by learning structural features observed in the event log. [17] presents an LSTM-based prediction model that uses the prefix and suffix sequences of events with missing activity labels as input to predict the missing labels, demonstrating high repair capability. [18] introduces the BERT4Log model and weak behavior profile theory, combined with a multi-layer multi-head attention mechanism, for interpretable repair of low-quality event logs. [19] proposes a convolutional neural network model that incorporates trace behavior features to repair missing activities in traces. The core idea is to transform the event log of a business process transition into spatial data based on the dimensions of time attributes and activity attributes, convert it into an image matrix, and train a convolutional neural network model to predict the missing activities.

2.2 Case-Level Repair

The existing techniques for case-level repair mainly focus on solving the problem 2 (denoted as ②) in Figure 1, where there exists at least one missing event in a case, along with multiple attributes of the given event are erroneous. A method combining random Petri nets, alignment, and Bayesian networks was proposed in [20] to recover missing activities and timestamps in event logs. [21] developed advanced indexing and pruning techniques to reduce the search space. [22] utilizes process decomposition techniques and heuristic methods to effectively prune unfeasible sub-processes that fail to produce minimal repairs. Both [21] and [22] aimed at minimizing the search space as much as possible to improve the efficiency of repairing events.

2.3 Summary of Existing Work

The attribute-level repair techniques are effective in repairing known anomalies or missing attributes; they are unable to handle cases where the missing information is unknown or when there are sequence anomalies and activity repetitions, as shown in problems 3 and 4 (denoted as ③ and ④) in Figure 1. On the other hand, case-level repair techniques rely heavily on process models and may not perform well in the absence of a process model. The specific comparison of existing techniques is shown in Table 1, where the symbol ✓ represents the scope of techniques considered. The specific meanings of the symbols are as follows:

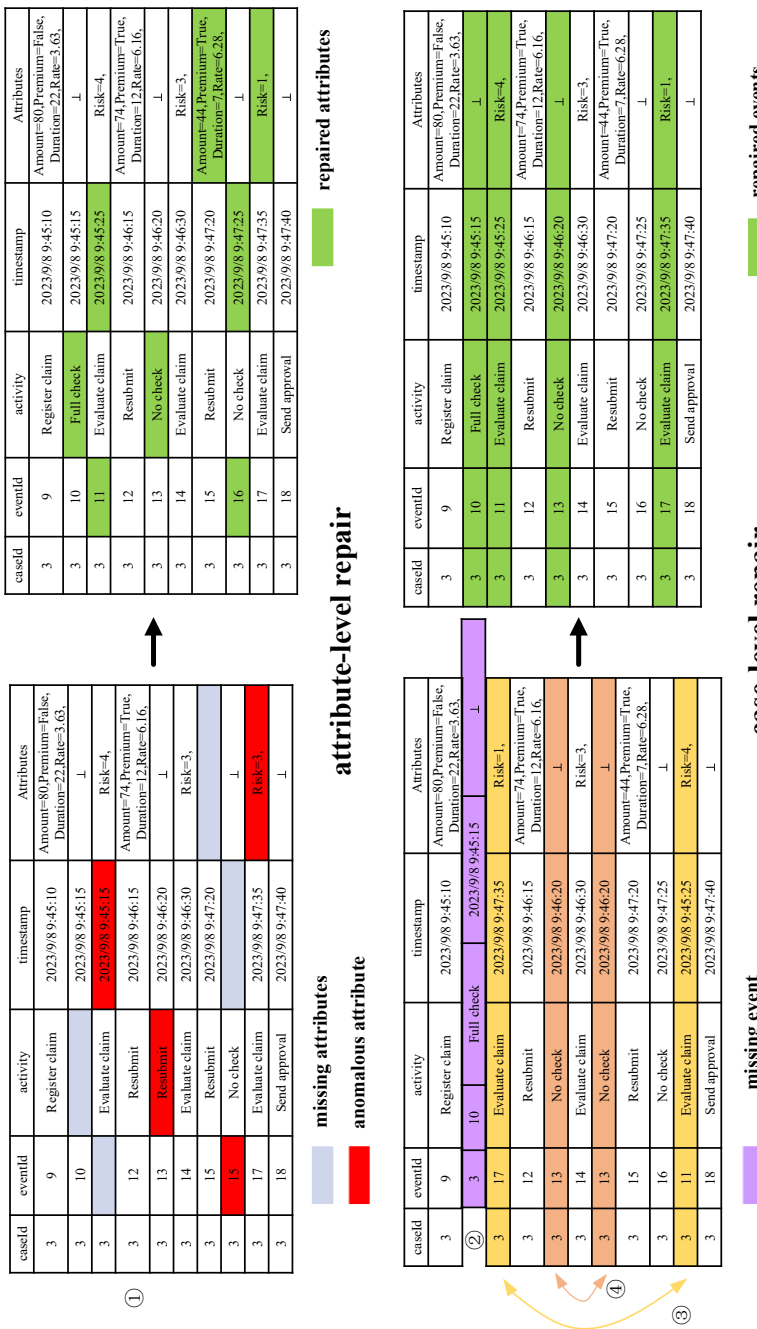


Figure 1. Log repair

F1: Deterministic repair for known anomalies or missing attributes, where the position of the anomaly or missing attribute is clearly identified.

F2: Recovery of a single attribute's missing values in a trace, mainly activity names.

F3: Recovery of multiple attributes' missing values in a trace.

F4: Incorporation of process models.

F5: Uncertain repair for missing attributes, where the position of the missing attribute is unknown.

F6: Uncertain repair, where the position of the missing attribute and the existence of event repetitions or sequence changes are unknown.

F7: Interpretable missing attribute repair.

Repair Type	Existing Technologies	F1	F2	F3	F4	F5	F6	F7
Attribute-level repair	[11, 14, 16, 17]	✓		✓				
	[12, 13, 19]	✓	✓					
	[15]						✓	
	[18]	✓	✓					✓
Case-level repair	[20, 21, 22]			✓	✓	✓		
The methods in this paper				✓		✓	✓	

Table 1. The comparison of existing techniques

Based on the aforementioned issues, this paper introduces the concept of an activity feature graph to detect and repair the mentioned problems. It utilizes activity feature graphs to compare abnormal behaviors with repaired behaviors, thereby identifying the causes of anomalies. Additionally, it enables further analysis of the impact of data. Furthermore, this paper also proposes the recovery of average behavior characteristics for missing values in the activity feature graph.

3 PRELIMINARIES

Definition 1 (Event e [23])**.** Let $\mathcal{E}$ be the event space, A be a set of activity, and AT be the set of all event attributes. An event $e = (\textit{case id}, \textit{event id}, \textit{activity}, \textit{timestamp}, \textit{Attributes})$ where $e \in \mathcal{E}$, is a quadruple. Here, *case id* is a case identifier that indicates the case or trace to which the event belongs, *event id* is the identifier of the event, serving as a unique identifier, $\textit{activity} \in A$ is the name of the activity, providing a simple representation of the event, $\textit{timestamp} \in AT$ represents the completion time of the event, and $\textit{Attributes} \in AT$ is a set of other attributes. Π is an attribute allocation function $\Pi : V \to U$, where V denotes the variables, i.e., attributes $at \in U$ generated during the process, and U denotes a set of variable values. The symbol $\perp \in U$ represents a null value, which does not necessarily indicate a missing value. Some events may temporarily lack assigned values for certain variables, or may have assigned values that are not recorded if they are not used in subsequent events, in which case they are replaced with $\perp$.

Definition 2 (Trace σ [23])**.** A trace $\sigma = \langle e_1, e_2, e_3, \ldots, e_n \rangle$ is a sequence of length n, where $\sigma \in \mathcal{E}^*$ and $\mathcal{E}^*$ represents the set of all sequences over $\mathcal{E}$. Here, $\sigma(i)$ denotes the i^{th} event, where $\Pi_{\text{case id}}(\sigma(i)) = \Pi_{\text{case id}}(\sigma(j))$ and $\Pi_{\text{timestamp}}(\sigma(i)) < \Pi_{\text{timestamp}}(\sigma(j))$ for $1 \leq i < j \leq n$. The notation $n = |\sigma|$ indicates the length of a trace.

Definition 3 (Event Log L [23])**.** An event log $L \in \mathcal{P}(\mathcal{E}^*)$ is a set of traces, where $\mathcal{P}$ represents the power set. Table 2 shows an example of an event log fragment generated from the process in Figure 2, where the rectangular boxes represent the generation functions of attributes for each event and the branching conditions of gateways.

CaseId	EventId	Activity	Timestamp	Attributes
3	9	Register claim	2023/9/8 9:45:10	Amount = 80, Premium = False, Duration = 22, Rate = 3.6
3	10	Full check	2023/9/8 9:45:15	$\perp$
3	11	Evaluate claim	2023/9/8 9:45:25	Risk = 4
3	12	Resubmit	2023/9/8 9:46:15	Amount = 74, Premium = True, Duration = 12, Rate = 6.16
3	13	No check	2023/9/8 9:46:20	$\perp$
3	14	Evaluate claim	2023/9/8 9:46:30	Risk = 3
3	15	Resubmit	2023/9/8 9:47:20	Amount = 44, Premium = True, Duration = 7, Rate = 6.28
3	16	No check	2023/9/8 9:47:25	$\perp$

Table 2. The fragment of event log

Definition 4 (Activity graph AG)**.** An activity graph is an ordered pair $AG = (A, D)$, where A is a set of nodes, which also represents a set of activities, and D is a set of direct succession relations, which are directed edges, $D = \{(a, a') \in A \times A | a = \Pi_{\text{activity}}(\sigma(i)) \wedge a' = \Pi_{\text{activity}}(\sigma(i+1)), i \in (1, n]\}$. Each trace in the event log corresponds to an activity graph. The activity graph in Figure 3 consists of blue activity nodes and a set of black directed edges, representing a trace. For convenience of representation, this paper replaces bidirectional relations represented by two inversely directed edges with a single bidirectional edge.

Definition 5 (Directed Adjacency Matrix of Activities DA)**.** The directed adjacency matrix DA represents the directed relationships between activity nodes. The activity graph $AG = (A, D)$ is a graph with m vertices, where m represents the number of activities in the event log ($|A| = m$). The specific expression of DA is

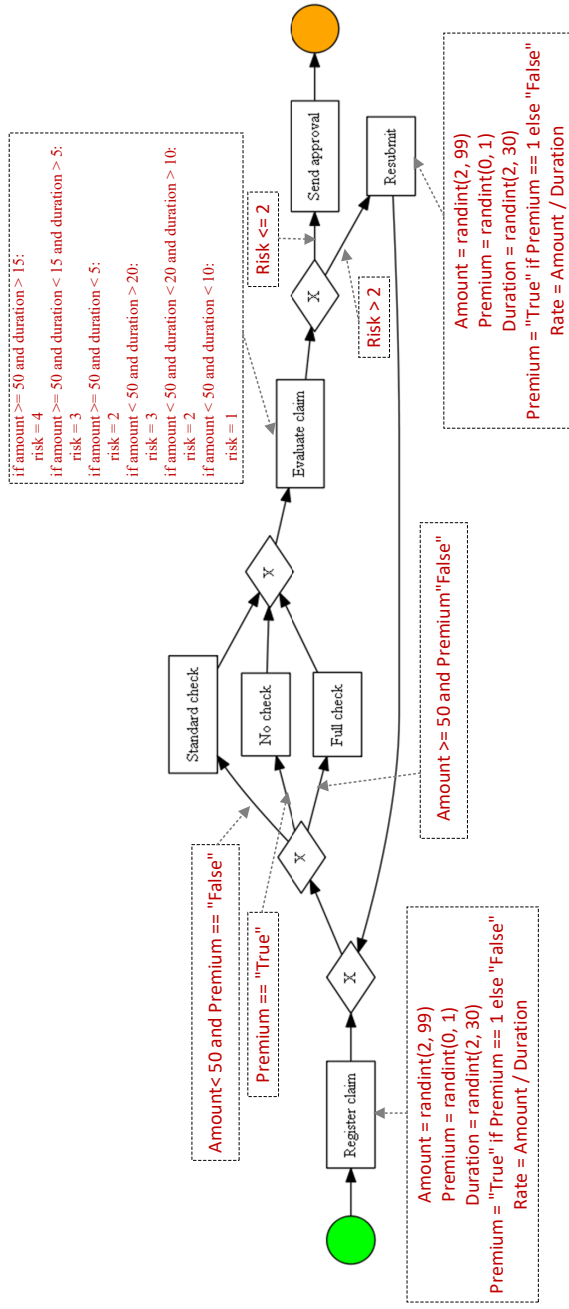


Figure 2. The process of the case

shown in Equation (1). The element $DA[i][j]$ represents the connection relationship between activity nodes i and j, where $i, j \in [1, m]$ represent the indices of the corresponding activities. If $DA[i][j] = 1$, it indicates that there is a directed edge from activity node i to j; if $DA[i][j] = 0$, it indicates that there is no directed edge from activity node i to j.

$$DA[i][j] = \begin{cases} 1, & \text{if } \langle a_i, a_j \rangle \in D, \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

The directed adjacency matrix in of activities Figure 3 visually represents the directed connections between activity nodes in the activity graph, which represents the control flow of a trace. The correspondence between the row number i of the adjacency matrix and activities is consistent. The row number representing activities in the adjacency matrix formed by different traces of the same log is consistent.

Definition 6 (Average Feature Matrix of Activities AF)**.** The average feature matrix is a matrix with $m \times d$ size, denoted as $AF_{m \times d} = [x_1, x_2, \ldots, x_m]^T$, where m represents the number of activity nodes, $d = |x_i|$ represents the length of the average feature vector for each node, and $i \in [1, m]$ denotes the index corresponding to the activity node. Each row x_i represents the average feature vector obtained from events with the same activity name in a trace, capturing the possible attribute values associated with that activity. The average feature vector x_i is calculated as follows in Equation (2).

$$avg\left(f\left(\Pi_{Attributes \wedge activity = A(i)}\left(\sigma(j)\right)\right)\right), \quad 1 \leq j \leq n, \quad n = |\sigma|. \tag{2}$$

Here, the functions avg and f represent taking the average value and obtaining the feature vector, respectively. Discrete values can be encoded using the one-hot method, while timestamp attribute can be transformed into continuous values by calculating time intervals, and the interval time is treated as the time spent attribute in the *Attributes* property. The average feature matrix of activities in Figure 3 records the average features of activity nodes in the activity graph.

Definition 7 (Activity Feature Graph AFG)**.** The activity feature graph $AFG = (AG, AF)$ is composed of the activity graph of a trace and the average feature matrix of activities for that trace. Here, AG represents the control flow of events within a trace, while AF represents the data changes associated with events within the trace.

The overall structure of the activity feature graph is illustrated in Figure 3, and it achieves behavioral analysis by integrating both control flow and data flow perspectives.

CNN (Convolutional Neural Network [24, 25]), GAT (Graph Attention Network [26, 27]), Decision tree [28], and LSTM (Long Short-Term Memory [29, 30]), as well as XGBoost (eXtreme Gradient Boosting [31, 32]), are common algorithms or models in the field of machine learning and artificial intelligence, all of which can

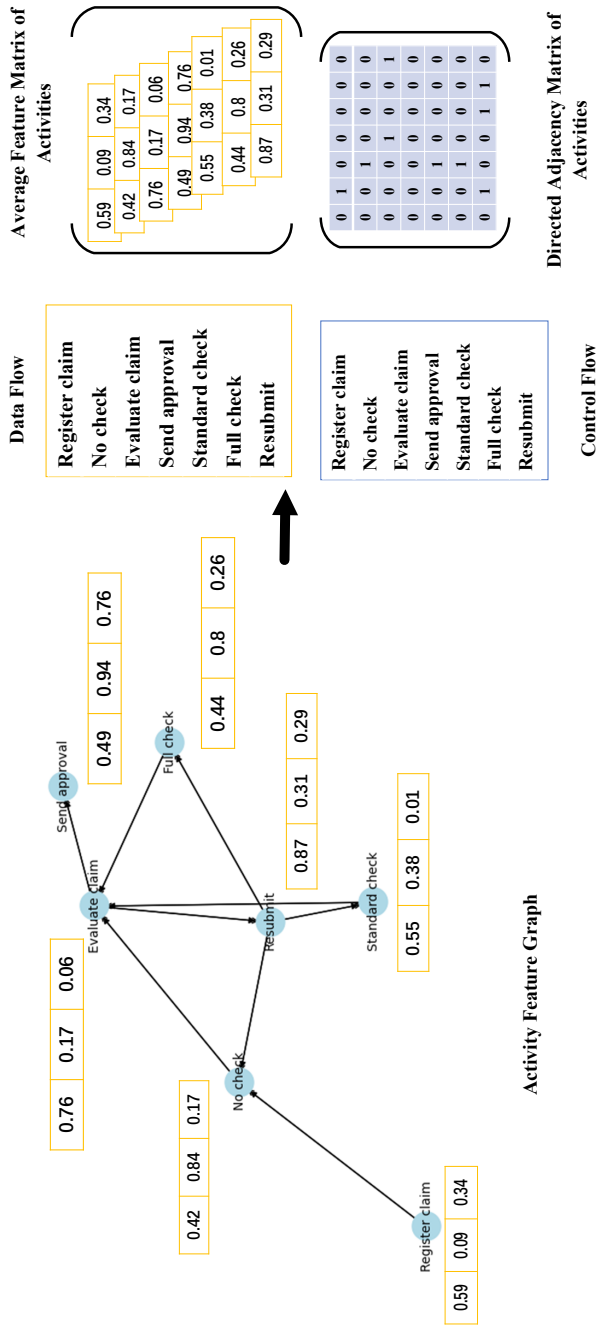


Figure 3. The activity feature graph

be used for classification and prediction tasks. Generally, CNNs extract features from images through convolutional layers and then reduce the dimension of feature maps through pooling layers. One or more fully connected layers are then used to map the extracted features to the final output categories. GAT introduces attention mechanisms based on GCN (Graph Convolutional Network [33, 34]). GCN is a classic graph neural network that uses fixed neighbor node weights for information propagation, where each neighbor node contributes equally to the target node. GAT, on the other hand, introduces attention mechanisms, allowing each node to focus on different degrees of neighboring nodes during information propagation. By transmitting and updating node features, GAT provides useful representations for graph data tasks, thus enabling various prediction and analysis tasks for graph data. A decision tree is a supervised learning method based on a tree-like structure. It starts from the root node to build a decision tree, dividing the dataset into subsets based on the chosen optimal features, and then repeats this process for each subset until the stopping conditions are met. The leaf nodes represent different class labels. LSTM is a type of recurrent neural network that excels at capturing time-dependent dependencies. Equipped with memory units and gate mechanisms, it can selectively retain and forget information from past time steps, making it highly effective in handling sequential and time series data. XGBoost is a gradient boosting algorithm that iteratively trains multiple weak learners and then combines them into a powerful model. Each weak learner focuses on correcting the prediction errors, gradually improving the overall model's performance.

KNNI (K-Nearest Neighbors Imputation [35, 36]), MICE (Multiple Imputation by Chained Equations [37, 38]), and GAIN (Generative Adversarial Imputation Nets [39]) are all methods used to handle missing data. KNNI is a nearest neighbor-based imputation method. For each missing value, KNNI searches for the k nearest neighbors (data points with known values) to estimate the missing value based on their known values, typically using Euclidean distance or other similarity measures to assess proximity. MICE is an iterative multiple imputation method. It models each missing value as a regression equation, using other variables with known values to predict the missing value. Each iteration updates one missing value until convergence or a predetermined number of iterations is reached, thus achieving the restoration of missing values. GAIN is a novel approach for missing data imputation that leverages the widely used GANs (Generative Adversarial Networks [40]) architecture to successfully perform data restoration even without complete data. The specific architecture of GAIN is illustrated in Figure 4 and consists of two neural networks: the generator and the discriminator. The generator aims to accurately impute the missing data, while the discriminator's objective is to distinguish between observed and imputed data. In Figure 4, the red portion in the original data represents the missing data. The mask matrix is a binary matrix where 1 represents observed data and 0 represents missing data, indicating which values in the input data are missing. The random matrix is used to generate random samples and introduce noise. The Hint Generator is also a generator model used to generate the hint matrix, which indicates which items are missing in the imputed data. When a value

in the hint matrix is 0.5, it indicates that the corresponding item may be missing in the imputed data, but the specific missingness is unknown. The estimated mask matrix represents the probability of correctly predicting the mask matrix M. The training process involves an adversarial process between the generator and discriminator, iterating until the generator successfully generates high-quality imputed data that can deceive the discriminator into believing it is observed data. This process continues until the generator produces imputed data that cannot be distinguished from the observed data by the discriminator.

4 SPECIFIC METHOD

The overall framework of the proposed method is depicted in Figure 5. Firstly, the target event log is preprocessed, which can include repair of event log integrity and feature encoding. Integrity repair can be achieved through attribute-level repair methods described in Section 2 to repair missing and abnormal attributes, or through commonly used data repair methods such as MICE, KNNI, etc., to obtain a relatively complete event log. This paper mainly focuses on feature encoding in preprocessing, which serves as the basis for subsequent work. Next, the preprocessed feature encoding is input into the anomaly detection model for anomaly detection. For the discovered anomalous behavior, it is input into the behavior repair model for repair, and the differences between the two are compared to indicate the location of the anomaly and the result of behavior repair. Then, the average feature vector of missing activity in the activity feature graph is repaired. Finally, behavior analysis is conducted through the activity feature graph.

4.1 Feature Encoding

In Figure 5, the preprocessing process marked as (1) provides three feature encoding methods proposed in this paper, which can be input into Graph Neural Networks, Convolutional Neural Networks, and ordinary deep learning or machine learning models, respectively. In the pre-process stage of Figure 5, two matrices of average feature matrix of activities and the directed adjacency matrix of activities are directly input into the Graph Neural Network (①); Then, the average feature matrix of activities and directed adjacency matrix of activities are stacked horizontally to form a two-dimensional image matrix (②), which serves as the input for the Convolutional Neural Network; After that, the two-dimensional matrix formed in ② is flattened into a one-dimensional vector (③), which is used as input for ordinary machine learning or neural networks.

The specific operations of the three encoding methods are shown in lines 13–21 of Algorithm 1. The mentioned three kinds of encoding methods can basically cover the inputs of existing learning models, enabling the input of activity feature graphs of traces into common models for training and learning, and subsequently using them for behavior analysis of each trace.

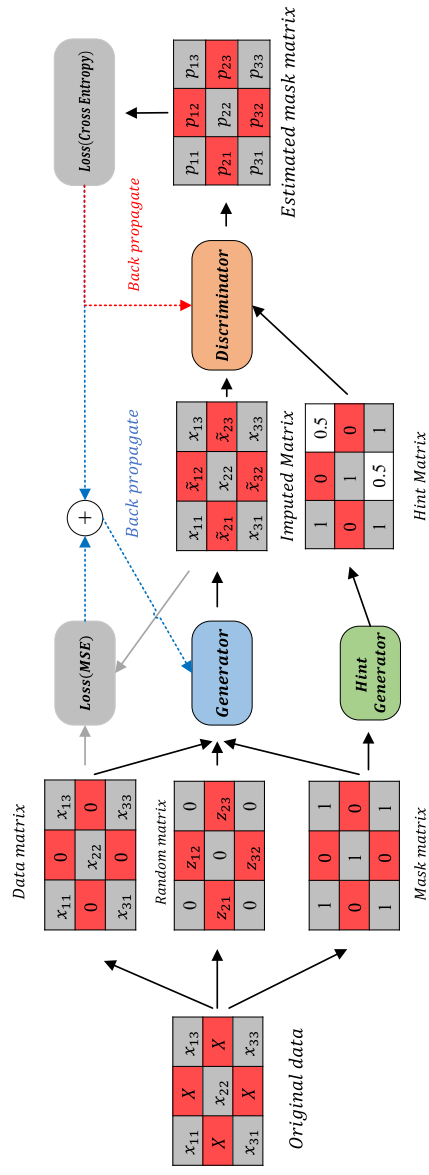


Figure 4. The architecture of the GAIN network

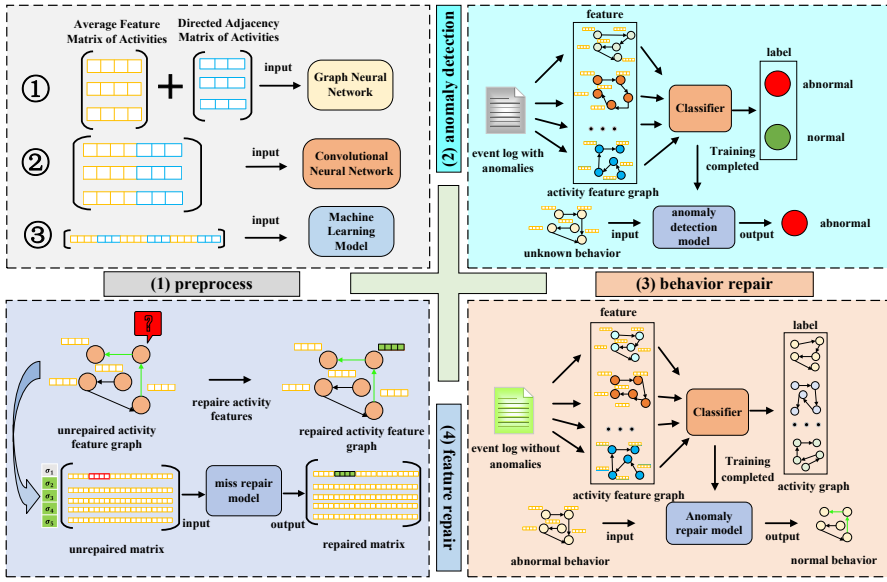


Figure 5. The overall framework

4.2 Anomaly Detection

Through the feature encoding, we can encode each trace in the event log as an activity feature graph, and input it into various learning models. The anomaly detection in Figure 5(2) illustrates the anomaly detection framework in this paper.

The event log contains both normal and abnormal behaviors, with labels assigned accordingly. If the event log only contains normal behavior, artificial injection of anomalies can be done to transform normal behavior into abnormal behavior and label it as such, as shown in lines 2–12 of Algorithm 1. The input of the classifier is the activity feature graph, and the output is the labels for abnormal and normal behavior. By training the classifier, it becomes capable of autonomously identifying abnormal and normal behavior in the given traces. Finally, the trained classifier serves as the anomaly detection model to predict unknown behavior as shown in lines 25–26 of Algorithm 1. The anomaly detection model is primarily a graph binary classification task, where each trace represents a graph with control flow and data flow content. By performing binary classification anomaly detection on the graph, it is possible to locate abnormal behavior.

4.3 Behavior Repairing

The behavior repair in Figure 5(3) presents the framework for repairing anomalous behaviors. The behavior repair task in this section is slightly different from the

Algorithm 1: Anomaly detection model training

Input: *encode* encoding method, Model model selection, *log* event log, *ratio* anomaly weight
Output: *model* trained anomaly detection model

foreach *trace in log* **do**
  $y \leftarrow$ random.choices($[0, 1], [ratio, 1 - ratio]$);
  if $y = 0$ **then**
    // 0 indicates anomaly;
    choice $\leftarrow$ random.choice($[0, 1, 2]$);
    if *choice* $= 0$ **then**
      trace $\leftarrow$ insert_duplicated_event_random(*trace*);
    else if *choice* $= 1$ **then**
      trace $\leftarrow$ delete_random_event(*trace*);
    else
      trace $\leftarrow$ randomly_swap_events(*trace*);
    end
  trace $\leftarrow$ get_average_activity_features(*trace*);
  $G \leftarrow$ convert_trace_to_activity_feature_graph(*trace*);
  if *encode* $=$ *'stacks'* **then**
    $g \leftarrow G$.feature_matrix $+ G$.adjacency_matrix;
  else if *encode* $=$ *'flatten'* **then**
    $g \leftarrow (G$.feature_matrix $+ G$.adjacency_matrix$)$.flatten;
  else
    $g \leftarrow G$;
  end
  data_x.append(g);
  data_y.append(y);
end
train_X, *test_X*, *train_y*, *test_y* $\leftarrow$ train_test_split(*data_x*, *data_y*, test_size, random_state);
model $\leftarrow$ Model().fit(train_X, train_y) // perform training;
return model;

anomaly detection and classification task. The training task of the behavior repair model focuses solely on predicting the classification of normal behavior traces. By training the model to differentiate different normal behaviors, it becomes possible to predict the specific normal behavior that a given abnormal behavior is most likely to belong to, thereby achieving the repair of anomalous behavior.

The repair procedures are presented as Algorithm 2. The input for the anomaly behavior repair classifier remains the activity feature graph, obtained exclusively from normal process traces, while the labels are activity graphs. This transforms the prediction of anomaly behavior repair into a multi-classification task on graphs,

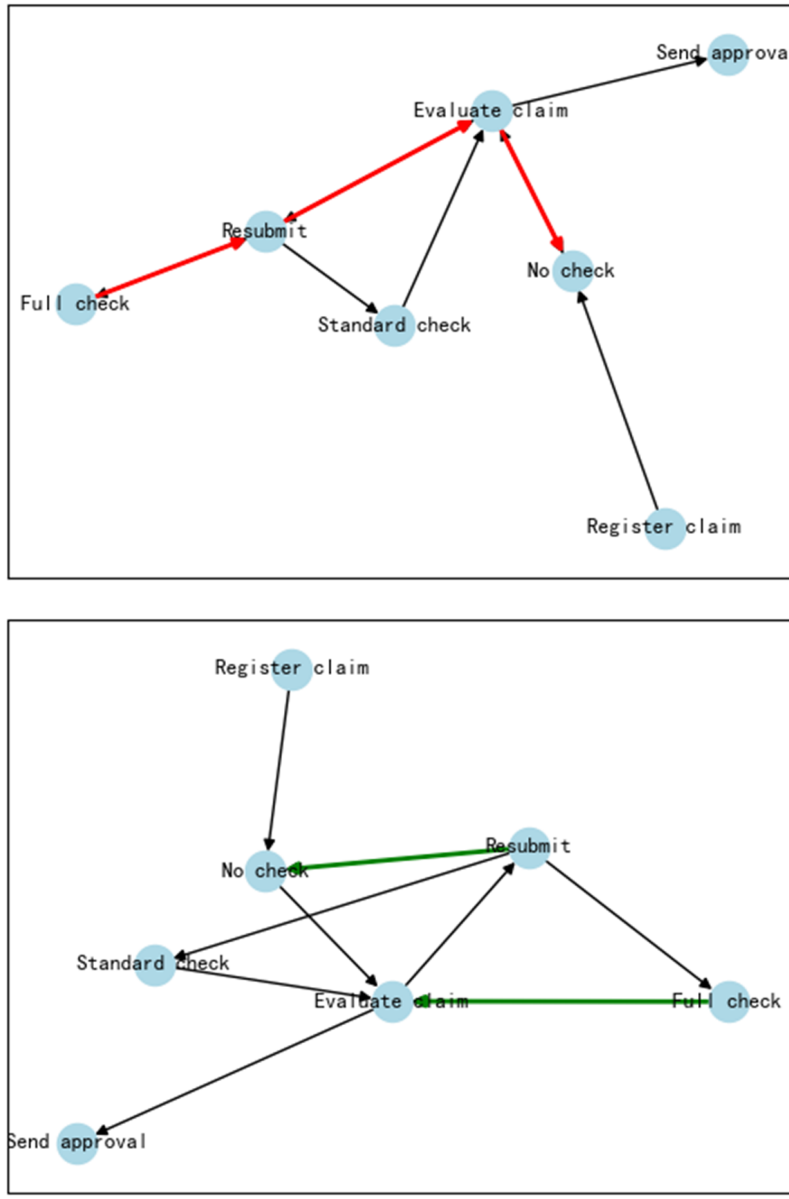


Figure 6. The anomaly localization

aiming to find the most similar normal behavior graph for each abnormal behavior graph. Finally, the trained classifier serves as the anomaly repair model. Figure 6 depicts the effect of anomaly behavior repair, where the left side represents the detected abnormal behavior graph, with red lines indicating the discovered anomaly, and the right side shows the repaired normal behavior, with green lines representing the repaired normal behavior. By comparing the differences between the two graphs, it is possible to detect and repair the anomalous behavior.

Algorithm 2: Anomaly repairing model training

Input: *log* event log
Output: *model* trained anomaly behavior repair model

```
G_label = {} //create a mapping from activity feature graph to label;
foreach trace in log do
    trace ← get_average_activity_features(trace);
    G ← convert_trace_to_activity_feature_graph(trace);
    if G_label.key doesn't have G.adjacency_matrix then
        G_label[G.adjacency_matrix] ← label;
        y ← label;
        label ← label + 1;
    else
        y ← G_label[G.adjacency_matrix];
    end
    data_x.append(G);
    data_y.append(y);
end
train_X, test_X, train_y, test_y ←
    train_test_split(data_x, data_y, test_size, random_state);
model ← Model().fit(train_X, train_y) // perform training;
return model;
```

4.4 Feature Repairing

The feature repair shown in Figure 5(4) presents the feature repair framework. Building on the behavior repair discussed in the previous section, we have identified the normal activity graph for the abnormal behavior. However, there may be cases where activity node attributes are missing, as indicated by the red question mark in Figure 5(4). In such scenarios, it is necessary to repair the missing average feature vectors for these nodes. Here, all the average feature vectors of nodes in the activity feature graph need to be concatenated into a trace vector. The average feature vectors of the missing nodes are all set to null values. Subsequently, together with the complete activity feature graph of the same activity graph, they form the unrepaired matrix. In Figure 5(4), the green traces in the unrepaired ma-

trix represent complete trace vectors, while the gray traces represent vectors with missing node attributes. Inputting this matrix into the repair model enables the generation of the repaired average feature vectors for the nodes. In this context, GAIN, KNNI, and MICE repair models can be utilized. These repair models do not require pre-training; they only need to take input matrices with missing vectors and can directly train on matrices with missing values. Ultimately, the repaired matrix is obtained, thereby achieving the repair of missing activity average feature vectors, as shown in Algorithm 3.

Algorithm 3: Missing node repair

Input: Model selection, *mG_sets* activity feature graphs collection with missing nodes, *log* event log
Output: *rG_sets* repaired activity feature graphs

```
cG_sets ← partition_log_by_activity_graph(log);
// partitioning event log into different sets of activity feature graphs based on the same activity graph;
iG_sets ← integrate_activity_feature_graphs(cG_sets, mG_sets);
// Merge the complete set of activity feature graphs with the set of // activity feature graphs to be repaired based on the activity graph;
foreach iG_set in iG_sets do
    // iterate the integrated set of activity feature graphs;
    umatrix ← get_unrepaired_matrix(rG_set);
    rmatrix ← Model().fit_transform(umatrix); // perform repairing;
    rG_set ← convert_matrix_to_activity_feature_graph(rmatrix);
    rG_sets.append(rG_set);
    // convert the repair matrix into activity feature graphs and add them to the set of repaired activity feature graphs;
end
return rG_sets;
```

5 EXPERIMENTAL EVALUATION

5.1 Experimental Setup

An overview of the experimental logs used in this paper is provided in Table 3. The Loan log[1] was generated using the process model and node functions as well as branch conditions presented in Figure 2. The *Resubmit* function in the log is identical to the *RegisterClaim* function, which increases the randomness of the log and introduces noise; The BPI_Challenge_2013_incidents log[2] illustrates the event

[1] https://github.com/zhangzhengzhuifeng/Activity-Feature-Graph/blob/main/Loan.zip

[2] https://doi.org/10.4121/uuid:500573e6-accc-4b0c-9576-aa5468b10cee

management process at Volvo IT; The Helpdesk log[3] depicts the ticketing management process of a software company in Italy; The BPIC12 log[4] presents the loan application process of a Dutch financial institution.

In our experiments, we assume that all of these logs are correct event logs and randomly selected process traces are subjected to three types of operations:

1. randomly inserting selected duplicate events after randomly selected events,
2. deleting randomly selected events, and
3. randomly swapping two events, in order to construct abnormal behaviors and conduct repair analysis.

Our experimental source codes and dataset are listed online[5].

Datasets	Trace Count	Event Count	Activity Count	Activity Graph Count
Loan	10 000	84 052	7	24
BPI_Challenge_2013_incidents	7 554	65 533	13	55
Helpdesk	4 580	21 348	14	172
BPIC12	13 087	262 200	24	1 736

Table 3. The introduction to Event Logs

In order to validate the effectiveness of our proposed method, we conduct three types of experiments. The first type of experiment utilizes two encoding methods and different classifiers to detect abnormal behaviors at various missing value ratios; The second type of experiment focuses on repairing abnormal behaviors, while the third type experiment employs different repair models to repair missing node attributes. We also use Friedman's test and Nemenyi post-hoc test to compare and analyze the experimental results, thereby demonstrating the effectiveness of our proposed method.

5.2 Analysis of Behavioral Anomaly Detection

Table 4 shows the accuracy results of the proposed method for anomaly detection, based on five different classifiers (named CNN, GAT, Decision tree, XGBoost and LSTM, respectively) on various datasets with injected anomaly ratios. In Table 4, CNN, GAT, and Decision Tree 1 use the encoding method proposed in this paper, while XGBoost, LSTM, and Decision Tree 2 use the prefix trace encoding method commonly used in process prediction in natural language processing, where the entire trace is considered as a prefix trace, and each event and its attributes within

[3] https://doi.org/10.4121/uuid:0c60edf1-6f83-4e75-9367-4c63b3e9d5bb

[4] https://doi.org/10.4121/uuid:3926db30-f712-4394-aebc-75976070e91f

[5] https://github.com/zhangzhengzhuifeng/Activity-Feature-Graph/blob/main/Loan.zip

Datasets	Anomaly Ratio	CNN	GAT	Decision Tree 1	XGBoost	LSTM	Decision Tree 2
Loan	5 %	98.95 %	96.40 %	99.50 %	99.20 %	95.25 %	99.40 %
	10 %	99.00 %	93.55 %	99.25 %	99.15 %	89.90 %	99.45 %
	15 %	98.30 %	86.30 %	98.95 %	99.30 %	84.85 %	99.25 %
	20 %	98.95 %	83.65 %	98.75 %	99.05 %	79.45 %	99.00 %
BPI_Challenge_2013_incidents	5 %	94.97 %	94.84 %	92.46 %	96.36 %	94.57 %	93.98 %
	10 %	92.72 %	90.14 %	87.56 %	94.51 %	90.40 %	91.07 %
	15 %	87.56 %	85.57 %	81.40 %	91.00 %	84.32 %	86.30 %
	20 %	86.23 %	81.73 %	79.81 %	90.07 %	81.14 %	85.37 %
Helpdesk	5 %	94.43 %	96.07 %	96.40 %	95.96 %	87.99 %	94.98 %
	10 %	92.14 %	88.54 %	94.43 %	92.47 %	84.28 %	90.94 %
	15 %	89.74 %	84.50 %	91.27 %	90.94 %	78.06 %	89.96 %
	20 %	88.21 %	81.33 %	89.85 %	89.96 %	71.29 %	85.04 %
BPIC12	5 %	97.21 %	94.65 %	97.75 %	98.05 %	95.15 %	96.75 %
	10 %	94.73 %	86.40 %	94.73 %	95.80 %	89.38 %	93.96 %
	15 %	94.04 %	85.60 %	94.19 %	95.45 %	85.94 %	91.41 %
	20 %	91.44 %	83.04 %	**92.44** %	92.09 %	79.22 %	89.23 %

Table 4. The accuracy of anomalous behavior detection on different classifiers

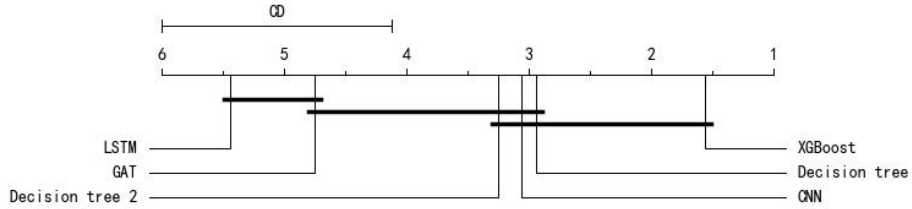


Figure 7. The Nemenyi Test for accuracy of anomaly detection on different methods

the prefix traces are treated as word features. Only one-hot encoding is applied to non-numeric attributes for both encoding methods.

The Friedman's test is conducted to examine the accuracy of different models on different datasets, yielding a p-value of 4.259×10^{-10}, which is much less than 0.05, indicating significant differences among different models. Figure 7 presents the results of the Nemenyi post-hoc test regarding the accuracy of different methods in anomaly detection. It can be observed that the XGBoost method using prefix trace encoding performs the best, but there is no significant difference compared to Decision tree 1 and CNN models using the encoding method proposed in this paper. Additionally, it is evident that the encoding method proposed in this paper is not inferior to the prefix trace encoding in terms of anomaly behavior detection, from the comparison between Decision tree 1 and Decision tree 2 using the same model.

Furthermore, it is worth noting that the deep learning models require the deep consideration of network structure and parameter tuning. In this paper, only the

most basic model framework was used, which may have contributed to the relatively high accuracy of the Decision Tree model. The prefix trace encoding records every position of the event that occurs, while the proposed encoding method retains only the connections between activities and the average attributes of events, effectively compressing the representation of the prefix trace. However, its ability to detect anomalies is not weaker than prefix trace encoding.

5.3 Analysis of Activity Graph Repair

Table 5 presents the results of the repair success rate obtained by directly using normal behavior as the training set and anomalous behavior as the test set after injecting anomalies into different datasets. The repair success rate is calculated according to Equation (3), where n represents the number of anomalous traces and m represents the number of predicted original activity graphs. Similar to the anomaly detection experiments, the predicted labels here are activity graphs.

$$Success\ Rate = \frac{m}{n}. \tag{3}$$

The Friedman's test was conducted with a p-value of 2.240×10^{-8}, which is significantly less than 0.05, indicating significant differences among mentioned methods. The Nemenyi post-hoc test results, as shown in Figure 8, reveal that the results of CNN, GAT, and Decision tree 1 are significantly better than those of XGBoost, LSTM, and Decision tree 2. This confirms the effectiveness of the trace encoding method proposed in this paper. While prefix trace encoding can accurately record the occurrence positions of events, any missing or repeated events within the trace may lead to inconsistencies in the original order, thus preventing successful repair. However, the encoding method proposed in this paper retains the original information as much as possible, enabling successful repair.

However, in the Helpdesk dataset, the highest accuracy achieved is only slightly above 50 %. This may be due to the presence of anomalies in the real-world dataset. This paper assumes that the real-world dataset is the normal dataset by default. Even in this scenario, the proposed method achieves over 70 % accuracy on other real-world datasets, demonstrating its effectiveness.

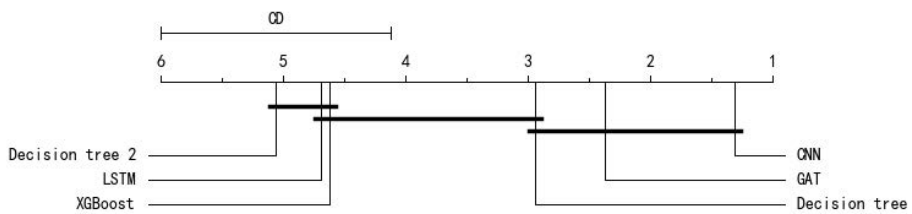


Figure 8. The Nemenyi Test for success rate of anomalous behavior repair on different methods

Datasets	Anomaly Ratio	CNN	GAT	Decision Tree 1	XGBoost	LSTM	Decision Tree 2
Loan	5 %	93.31 %	81.54 %	89.05 %	58.82 %	48.88 %	53.14 %
	10 %	92.61 %	81.72 %	89.01 %	63.35 %	58.62 %	65.91 %
	15 %	93.36 %	79.47 %	88.39 %	58.39 %	38.69 %	56.80 %
	20 %	93.86 %	78.84 %	88.16 %	60.80 %	67.59 %	61.25 %
BPI_Challenge_2013_incidents	5 %	83.15 %	69.34 %	63.26 %	66.67 %	32.28 %	53.17 %
	10 %	77.95 %	70.87 %	62.34 %	67.25 %	35.98 %	54.72 %
	15 %	81.72 %	71.76 %	72.02 %	67.20 %	37.52 %	56.24 %
	20 %	77.94 %	67.90 %	69.60 %	66.43 %	38.17 %	52.10 %
Helpdesk	5 %	42.86 %	46.43 %	13.39 %	15.63 %	43.30 %	7.14 %
	10 %	50.50 %	56.06 %	12.72 %	16.50 %	44.53 %	7.55 %
	15 %	50.31 %	52.31 %	16.77 %	16.15 %	46.00 %	6.62 %
	20 %	53.71 %	57.31 %	15.50 %	15.94 %	51.64 %	6.66 %
BPIC12	5 %	75.19 %	43.07 %	70.47 %	20.09 %	22.98 %	42.31 %
	10 %	76.04 %	43.05 %	58.07 %	18.45 %	26.44 %	39.70 %
	15 %	75.63 %	43.51 %	73.06 %	19.91 %	26.48 %	21.45 %
	20 %	76.68 %	49.71 %	72.74 %	18.33 %	25.51 %	20.41 %

Table 5. The repair success rate of anomalous behavior on different classifiers

5.4 Analysis of Node Attribute Repair

Table 6 presents the RSME (The Root Mean Square Error) obtained by applying the GAIN, KNNI, and MICE models. These models are used to directly repair the missing matrix formed by randomly setting missing proportions on the average feature vectors of activities in the event log. The whole event log contains the maximum number of feature graphs extracted. The lower the RSME value, the better the repair performance.

Under these circumstances, the repair of the missing matrix does not need to consider the sequential relationships between activities, because they all belong to the same type of activity graph, and the average feature vectors of each node's activities are arranged in a certain order.

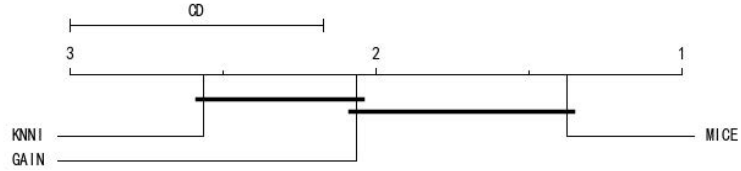


Figure 9. The Nemenyi Test for RSME of node attribute repair on different methods

The Friedman's test was conducted on Table 6, yielding a p-value of 0.003388, which is less than 0.05. This led to the Nemenyi post-hoc test results shown in Figure 9. Among the mentioned methods, MICE performs better, while there is

Datasets	Missing Ratio	GAIN	KNNI	MICE
Loan	5 %	0.432445	0.344897	0.325893
	10 %	0.406709	0.32165	0.289357
	15 %	0.399479	0.3016	0.280275
	20 %	0.423272	0.34735	0.310561
BPI Challenge 2013 incidents	5 %	0.392417	0.769872	0.352725
	10 %	0.416584	0.738547	0.411478
	15 %	0.428099	0.718578	0.344023
	20 %	0.487208	0.702862	0.370748
Helpdesk	5 %	0.306925	0.301844	1.021869
	10 %	0.315487	0.409658	0.480837
	15 %	0.321282	0.523824	0.481666
	20 %	0.319518	0.551897	0.403967
BPIC12	5 %	0.104823	0.194868	0.069976
	10 %	0.118687	0.202375	0.030021
	15 %	0.122232	0.223503	0.069627
	20 %	0.121775	0.192945	0.062202

Table 6. The RSME of node attribute repair on different classifiers

no significant difference between the GAIN method and the other two methods. However, it should be noted that this paper focuses on repairing the average feature vectors of activities, which represent the average values of activity features that occur multiple times within a trace, while the individual feature vectors for each occurrence are missing.

6 CONCLUSION

This paper proposes a method for detecting and repairing behavioral anomalies that may occur in traces of event logs, such as missing events, repeated events, and swapped event sequences. To achieve this, the concept of activity feature graphs is introduced, which combines the perspectives of control flow and data flow. Three encoding methods are provided to train and learn different classifiers. The proposed method can locate abnormal behaviors, identify the correct behaviors, and repair missing node attributes in activity feature graphs. Additionally, a series of experiments is conducted to validate the effectiveness of the proposed method.

Moreover, it is worth noting that, as activity feature graphs may have different paths to traverse, the proposed repair method could randomly select the most common path and fill it in the log, which is a potential limitation of the proposed method. Hence, these operations may deviate significantly from the original event flow, and the event attributes are given based on the average feature attributes of activities, not a specific one. Nonetheless, for individual behaviors, they may also be analyzed using the activity feature graph approach, which requires specific selection by relevant personnel.

In the forthcoming studies, we will further conduct experiments to compare our method with those of large language models (LLMs). As LLMs need a corpus to support, it is a complicated issue to generate an event log corpus or traces corpus, which has not been resolved currently.

Acknowledgments

This work was supported by the Anhui Provincial Funding Program for Domestic Visiting and Training of Young Backbone Teachers (No. JNFX2025025).

REFERENCES

[1] van der Aalst, W.: Process Mining: Data Science in Action. Springer, 2016, doi: 10.1007/978-3-662-49851-4.

[2] Chapela-Campa, D.—Mucientes, M.—Lama, M.: Understanding Complex Process Models by Abstracting Infrequent Behavior. Future Generation Computer Systems, Vol. 113, 2020, pp. 428–440, doi: 10.1016/j.future.2020.07.030.

[3] van der Aalst, W.—Weijters, T.—Maruster, L.: Workflow Mining: Discovering Process Models from Event Logs. IEEE Transactions on Knowledge and Data Engineering, Vol. 16, 2004, No. 9, pp. 1128–1142, doi: 10.1109/TKDE.2004.47.

[4] Günther, C. W.—van der Aalst, W. M. P.: Fuzzy Mining – Adaptive Process Simplification Based on Multi-Perspective Metrics. In: Alonso, G., Dadam, P., Rosemann, M. (Eds.): Business Process Management (BPM 2007). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 4714, 2007, pp. 328–343, doi: 10.1007/978-3-540-75183-0_24.

[5] Weijters, A. J. M. M.—Ribeiro, J. T. S.: Flexible Heuristics Miner (FHM). 2011 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2011, pp. 310–317, doi: 10.1109/CIDM.2011.5949453.

[6] Leemans, S. J. J.—Fahland, D.—van der Aalst, W. M. P.: Discovering Block-Structured Process Models from Event Logs Containing Infrequent Behaviour. In: Lohmann, N., Song, M., Wohed, P. (Eds.): Business Process Management Workshops (BPM 2013). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 171, 2014, pp. 66–78, doi: 10.1007/978-3-319-06257-0_6.

[7] van der Aalst, W.—Adriansyah, A.—de Medeiros, A. K. A.—Arcieri, F.—Baier, T. et al.: Process Mining Manifesto. In: Daniel, F., Barkaoui, K., Dustdar, S. (Eds.): Business Process Management Workshops (BPM 2011). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 99, 2012, pp. 169–194, doi: 10.1007/978-3-642-28108-2_19.

[8] van Eck, M. L.—Lu, X.—Leemans, S. J. J.—van der Aalst, W. M. P.: PM2: A Process Mining Project Methodology.

[9] Huang, R.—Wang, J.—Song, S.—Lin, X.—Zhu, X.—Pei, J.: Efficiently Cleaning Structured Event Logs: A Graph Repair Approach. ACM Transactions on Database System (TODS), Vol. 48, 2023, No. 1, Art. No. 3, doi: 10.1145/3571281.

[10] Bose, R. P. J. C.—Mans, R. S.—van der Aalst, W. M. P.: Wanna Improve Process Mining Results? 2013 IEEE Symposium on Computational Intelligence and Data Mining (CIDM), 2013, pp. 127–134, doi: 10.1109/CIDM.2013.6597227.

[11] Nguyen, H. T. C.—Lee, S.—Kim, J.—Ko, J.—Comuzzi, M.: Autoencoders for Improving Quality of Process Event Logs. Expert Systems with Applications, Vol. 131, 2019, pp. 132–147, doi: 10.1016/j.eswa.2019.04.052.

[12] Xu, J.—Liu, J.: A Profile Clustering Based Event Logs Repairing Approach for Process Mining. IEEE Access, Vol. 7, 2019, pp. 17872–17881, doi: 10.1109/ACCESS.2019.2894905.

[13] Liu, J.—Xu, J.—Zhang, R.—Reiff-Marganiec, S.: A Repairing Missing Activities Approach with Succession Relation for Event Logs. Knowledge and Information Systems, Vol. 63, 2021, No. 2, pp. 477–495, doi: 10.1007/s10115-020-01524-6.

[14] Sim, S.—Bae, H.—Choi, Y.: Likelihood-Based Multiple Imputation by Event Chain Methodology for Repair of Imperfect Event Logs with Missing Data. 2019 International Conference on Process Mining (ICPM), IEEE, 2019, pp. 9–16, doi: 10.1109/ICPM.2019.00013.

[15] Horita, H.—Kurihashi, Y.—Miyamori, N.: Extraction of Missing Tendency Using Decision Tree Learning in Business Process Event Log. Data, Vol. 5, 2020, No. 3, Art. No. 82, doi: 10.3390/data5030082.

[16] Sim, S.—Bae, H.—Liu, L.: Bagging Recurrent Event Imputation for Repair of Imperfect Event Log with Missing Categorical Events. IEEE Transactions on Services Computing, Vol. 16, 2021, No. 1, pp. 108–121, doi: 10.1109/TSC.2021.3118381.

[17] Lu, Y.—Chen, Q.—Poon, S. K.: A Deep Learning Approach for Repairing Missing Activity Labels in Event Logs for Process Mining. Information, Vol. 13, 2022, No. 5, Art. No. 234, doi: 10.3390/info13050234.

[18] Li, B.—Fang, H.—Mei, Z.: Interpretable Repair Method for Event Logs Based on BERT and Weak Behavioral Profiles. Computer Science, Vol. 50, 2023, No. 5, pp. 38–51, doi: 10.11896/jsjkx.220900030 (in Chinese).

[19] Liu, W.—Fang, H.—Zhang, S.: Missing Activity Log Repair Method Based on Image Data Using CNN. Computer Integrated Manufacturing Systems, Vol. 30, 2024, No. 8, pp. 2787–2796 (in Chinese).

[20] Rogge-Solti, A.—Mans, R. S.—van der Aalst, W. M. P.—Weske, M.: Improving Documentation by Repairing Event Logs. In: Grabis, J., Kirikova, M., Zdravkovic, J., Stirna, J. (Eds.): The Practice of Enterprise Modeling (PoEM 2013). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 165, 2013, pp. 129–144, doi: 10.1007/978-3-642-41641-5_10.

[21] Wang, J.—Song, S.—Zhu, X.—Lin, X.: Efficient Recovery of Missing Events. Proceedings of the VLDB Endowment, Vol. 6, 2013, No. 10, pp. 841–852, doi: 10.14778/2536206.2536212.

[22] Song, W.—Xia, X.—Jacobsen, H. A.—Zhang, P.—Hu, H.: Heuristic Recovery of Missing Events in Process Logs. 2015 IEEE International Conference on Web Services, 2015, pp. 105–112, doi: 10.1109/ICWS.2015.24.

[23] Bazhenova, E.—Buelow, S.—Weske, M.: Discovering Decision Models from Event Logs. In: Abramowicz, W., Alt, R., Franczyk, B. (Eds.): Business Informa-

tion Systems (BIS 2016). Springer, Cham, Lecture Notes in Business Information Processing, Vol. 255, 2016, pp. 237–251, doi: 10.1007/978-3-319-39426-8_19.

[24] Krizhevsky, A.—Sutskever, I.—Hinton, G. E.: ImageNet Classification with Deep Convolutional Neural Networks. Communications of the ACM, Vol. 60, 2017, No. 6, pp. 84–90, doi: 10.1145/3065386.

[25] Hu, X.—Qiu, G.—Karimi, H. R.—Zhang, D.: TFF-CNN: Distributed Optical Fiber Sensing Intrusion Detection Framework Based on Two-Dimensional Multi-Features. Neurocomputing, Vol. 564, 2024, Art. No. 126959, doi: 10.1016/j.neucom.2023.126959.

[26] Veličkovic, P.—Cucurull, G.—Casanova, A.—Romero, A.—Liò, P.—Bengio, Y.: Graph Attention Networks. CoRR, 2017, doi: 10.48550/arXiv.1710.10903.

[27] Xu, F.—Zeng, L.—Huang, Q.—Yan, K.—Wang, M.—Sheng, V. S.: Hierarchical Graph Attention Networks for Multi-Modal Rumor Detection on Social Media. Neurocomputing, Vol. 569, 2024, Art. No. 127112, doi: 10.1016/j.neucom.2023.127112.

[28] Loh, W. Y.: Classification and Regression Trees. WIREs Data Mining and Knowledge Discovery, Vol. 1, 2011, No. 1, pp. 14–23, doi: 10.1002/widm.8.

[29] Hochreiter, S.—Schmidhuber, J.: Long Short-Term Memory. Neural Computation, Vol. 9, 1997, No. 8, pp. 1735–1780, doi: 10.1162/neco.1997.9.8.1735.

[30] Liu, H.—Yang, J.—Chang, C. H.—Wang, W.—Zheng, H. T.—Jiang, Y.—Wang, H.—Xie, R.—Wu, W.: AOG-LSTM: An Adaptive Attention Neural Network for Visual Storytelling. Neurocomputing, Vol. 552, 2023, Art. No. 126486, doi: 10.1016/j.neucom.2023.126486.

[31] Chen, T.—Guestrin, C.: XGBoost: A Scalable Tree Boosting System. Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD '16), 2016, pp. 785–794, doi: 10.1145/2939672.2939785.

[32] Yan, Z.—Wang, J.—Sheng, L.—Yang, Z.: An Effective Compression Algorithm for Real-Time Transmission Data Using Predictive Coding with Mixed Models of LSTM and XGBoost. Neurocomputing, Vol. 462, 2021, pp. 247–259, doi: 10.1016/j.neucom.2021.07.071.

[33] Kipf, T. N.—Welling, M.: Semi-Supervised Classification with Graph Convolutional Networks. CoRR, 2016, doi: 10.48550/arXiv.1609.02907.

[34] Zeng, D.—Zha, E.—Kuang, J.—Shen, Y.: Multi-Label Text Classification Based on Semantic-Sensitive Graph Convolutional Network. Knowledge-Based Systems, Vol. 284, 2024, 111303 pp., doi: 10.1016/j.knosys.2023.111303.

[35] Van Hulse, J.—Khoshgoftaar, T. M.: Incomplete-Case Nearest Neighbor Imputation in Software Measurement Data. Information Sciences, Vol. 259, 2014, pp. 596–610, doi: 10.1016/j.ins.2010.12.017.

[36] Sivakumar, J.—Ramamurthy, K.—Radhakrishnan, M.—Won, D.: Synthetic Sampling from Small Datasets: A Modified Mega-Trend Diffusion Approach Using k-Nearest Neighbors. Knowledge-Based Systems, Vol. 236, 2022, Art. No. 107687, doi: 10.1016/j.knosys.2021.107687.

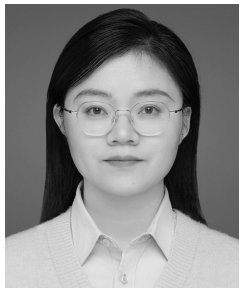
[37] Demirtas, H.: Flexible Imputation of Missing Data. Journal of Statistical Software, Vol. 85, 2018, Book Review 4, doi: 10.18637/jss.v085.b04.

[38] Samad, M. D.—Abrar, S.—Diawara, N.: Missing Value Estimation Using Clustering and Deep Learning Within Multiple Imputation Framework. Knowledge-Based Systems, Vol. 249, 2022, Art. No. 108968, doi: 10.1016/j.knosys.2022.108968.

[39] Yoon, J.—Jordon, J.—Schaar, M.: GAIN: Missing Data Imputation Using Generative Adversarial Nets. In: Dy, J., Krause, A. (Eds.): Proceedings of the 35th International Conference on Machine Learning. Proceedings of Machine Learning Research (PMLR), Vol. 80, 2018, pp. 5689–5698, http://proceedings.mlr.press/v80/yoon18a/yoon18a.pdf.

[40] Wang, K.—Gou, C.—Duan, Y.—Lin, Y.—Zheng, X.—Wang, F. Y.: Generative Adversarial Networks: Introduction and Outlook. IEEE/CAA Journal of Automatica Sinica, Vol. 4, 2017, No. 4, pp. 588–598, doi: 10.1109/JAS.2017.7510583.

Lu Li received her B.Sc. degree in information and computing from the Anhui University of Science and Technology, China, in 2003 and her M.Sc. degree in computer science and engineering from Hefei University of Technology, China, in 2006. Her research interests include data mining, algorithm design and digital image processing.



Huan Fang received her B.Sc. degree in information and computing from the Anhui University of Science and Technology, China, in 2003 and her M.Sc. degree in computer science and engineering from the Shandong University of Science and Technology, China, in 2006 and her Ph.D. degree in computer science and engineering from the Hefei University of Technology, China, in 2013. From 2018 to 2021, she was a postdoc with the Anhui University of Science and Technology, Huainan, China. Her research interest includes the Petri nets, process mining, change mining and intelligent control. She is Professor with School of Mathematics and Big Data in Anhui University of Science and Technology. Her awards and honors include the Excellent Teachers of Anhui Province, China.

Shouzheng Zhang received his M.Sc. degree in information security from the Anhui University of Science and Technology, China, in 2006. His research interests include process mining and graph neural networks.