Variational Autoencoder and Graph Attention Root Cause Localization Model Based on Log Data and Graph Structure

Jianli Ding[1], Yanan Yan[1(✉)], Jing Wang[1(✉)], and Tantan Chen[2]

[1] Civil Aviation University of China, Tianjin, China
nivy.yan@foxmail.com, j_wang@cauc.edu.cn
[2] Accounting Centre of China Aviation, Beijing, China

Abstract. When conducting root cause localization, converting data into graph structures for feature extraction can represent complex dependency relationships among data more comprehensively and deeply. A root cause localization model (Log-based Graph Root Cause Localization, LGRCL) based on log data and graph structure is proposed. Process and group multi-source, unstructured log data according to root cause location problem scenarios, and build a log dependency graph. Secondly, in order to more accurately capture the dependencies between data, for feature extraction of log dependency graphs, Variational Auto Encoder (VAE) is used to capture the potential features of the graph structure, and graph attention is introduced to focus on key nodes in the graph. The proposed LGRCL model is compared with the baseline model and better adapts to the complexity of log data and graph structure. The accuracy of the LGRCL model in locating the top five root causes is as high as 94.71%. The accuracy is 35.36% higher than the baseline model on average. The root cause location efficiency is 13.71% higher on average than the baseline model.

Keywords: Log Data · Graph Structure · Log Dependency Graph · Feature Extraction · Root Cause Location

1 Introduction

The rapid development and widespread application of information technology have made systems and network environments of various sizes increasingly complex. What follows is an increase in system failures and performance problems. When a system failure occurs, rapid root cause location and feedback to engineers with a clear list of root causes to take measures are key steps to ensure system availability and performance.

Current root cause location mainly uses data such as logs [1–4] and KPI indicators [5, 6]. The results of these methods in locating the root cause of a fault are often not specific, such as attributing the problem to a certain component of the system. It is not possible to indicate exactly which aspect of the component caused the failure. Some event-based methods usually define events at the application level or module level [7,

D.-S. Huang et al. (Eds.): ICIC 2024, LNAI 14878, pp. 17–29, 2024.
https://doi.org/10.1007/978-981-97-5672-8_2

8], which also have the limitation of unspecific positioning granularity. Therefore, when engineers get the root cause list, they cannot take immediate measures.

MicroRCA [9], Wu et al. [10], Sieve [11] and Brandón et al. [12] automatically construct a topological diagram of the application and then analyze possible root causes on the diagram. DLA [13] also utilizes topological graphs to identify KPIs that may lead to observed anomalies. The establishment and maintenance of topology diagrams are full of complexities. For example, distributed systems and microservices often contain multiple service entities, which may be distributed on different physical or virtual machines, or even span multiple data centers.

In addition, methods based on KPI data [9, 14] also have problems in practical applications. Monitoring KPIs requires advanced configuration, and since KPI data may contain sensitive information about system performance and operating status, so access to KPI data may be limited when it comes to privacy and security issues.

In response to the above problems, this paper proposes a LGRCL model for root cause localization based on logs. Because log data usually exists in text form, and its format is relatively flexible. Almost all software and hardware generate logs. Therefore, it becomes relatively easy to collect, store and analyze logs in different environments. LGRCL first converts the log data into a graph structure, and the Log Dependency Graph (LDG) is proposed. In the process of constructing the graph, represent the nodes in the graph as a combination of components and log events, so that the final root cause list fed back to engineers will be at a specific operational granularity. Secondly, in order to better capture the relationships and characteristics between log events, the Variational Autoencoder and Graph Attention (VAEGA) method is proposed to extract features from the LDG. The VAEGA method combines VAE and graph attention network (GAT), and VAE can better capture the latent characteristics of the graph structure and take uncertainty into account when generating node representations. The graph attention mechanism enables the overall feature extraction module to more accurately capture the correlation and topological structure information between nodes when generating node features. This design enables the model to better distinguish the characteristics of different nodes during the learning process and better adapt to different network environments and system states, thereby improving the modeling ability of the system state.

2 System Log Processing and Log Dependency Graph

2.1 System Log Processing

When a system failure occurs, the log records all runtime information to facilitate root cause location. Each log includes timestamp, event type (log event or log template) and log parameters. The log template can be obtained by parsing the log, as shown in Fig. 1, each log template corresponds to a unique EventID, and each EventID contains several log messages. The log messages are distinguished by different parameters. In order to adapt to the problem scenario of root cause location, we refer to time series and content similarity, as well as contextual analysis of abnormal logs to more comprehensively and accurately capture the correlation between log events to group the logs. Each group is a collection of all logs involved in an event process to analyze the root cause of the

failure. Then, we generate an aggregated value for each EventID by representing the log template as a word embedding vector and introducing the binary feature of whether the log is faulty.

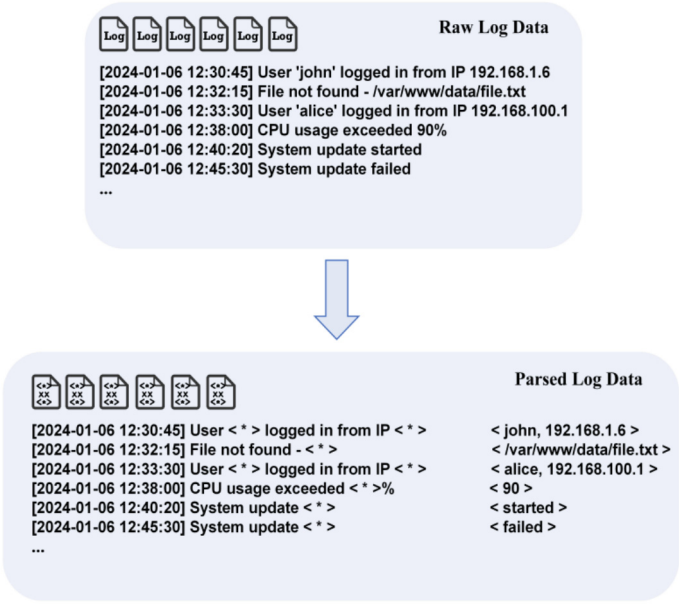

Fig. 1. Log parsing process

2.2 Log Dependency Graph

We consider using an undirected graph with attributes, denoted as $\mathcal{G} = (\mathcal{V}, \mathcal{E}, \mathrm{X})$, where $\mathcal{V} = \{v_1, ..., v_{|\mathcal{V}|}\}$ represents the node set and $\mathcal{E} = \{\rceil_1, ..., \rceil_{|\mathcal{E}|}\} \subseteq \mathcal{V} \times \mathcal{V}$ represents the edge set. If $(v_i, v_j) \in \mathcal{E}$, it means that there is an edge between node i and node j. $\mathrm{X} \in \mathbb{R}^{|\mathcal{V}| \times d}$ is the node attribute matrix, the i-th row represents the attribute of node i, and d represents the number of attributes.

Similarly, $\mathcal{G}$ can be described as (A, X), where $\mathrm{A} \in \mathbb{R}^{|\mathcal{V}| \times |\mathcal{V}|}$ represents the adjacency matrix of the graph, $\mathrm{A}_{ij} = \mathbb{I}[(v_i, v_j) \in \mathcal{E}]$ represents whether there is an edge from node i to node j, and $i, j \in \{1, ..., |\mathcal{V}|\}$.

Given a set of log data, let $\mathcal{L} = \{L_1, ..., L_{|\mathcal{L}|}\}$ represent a unique set of log templates. We divide the log data into M log groups $\mathcal{Q} = \{q_1, ..., q_m, ..., q_M\}$, where $q_m = \{q_{m1}, ..., q_{mn}, ..., q_{mN}\}$ is the N log messages in each group of logs. For each set of logs q_m, we construct an undirected graph $\mathcal{G}_m = (\mathcal{V}_m, \mathcal{E}_m, \mathrm{X}_m)$ with attributes to represent the relationship between logs. Specifically, each node $v_i \in \mathcal{V}_m$ corresponds to a log templateL $\in \mathcal{L}$. In addition, we extract the components corresponding to each log from the log data, then each node $v_i \in \mathcal{V}_m$ is a combination of the component and the log template. Attribute $x_i \in \mathrm{X}_m$ represents the semantic embedding of log templatei. In this way, a set of LDG $\{\mathcal{G}_1, ..., \mathcal{G}_m, ..., \mathcal{G}_M\}$ is obtained.

3 Model Framework

In order to convert log data into a graph structure and locate root causes, this paper proposes the LGRCL model, as shown in Fig. 2. First, we parse the original log data and group the logs according to the process of an abnormal event to structure the log data. Then, an undirected LDG is constructed for each log grouping. The feature extraction stage is mainly implemented through VAEGA. VAE is used to learn the potential representation of data, and the introduced graph attention mechanism performs well in dealing with complex and interrelated data such as log data by focusing adaptively on key nodes in the graph. Finally, the node features are aggregated through multi-head GAT, and the aggregated values of the nodes are sorted through a double-layer dense network, and a root cause list is obtained and fed back to the engineer.

3.1 Construction of LDG

The construction of the LDG is a crucial step in root cause analysis. It forms an undirected graph by capturing the dependencies between log events, which provides the basis for subsequent feature extraction and feature aggregation. The following describes the construction process of the LDG in detail, including log parsing, log grouping, and node vector representation.

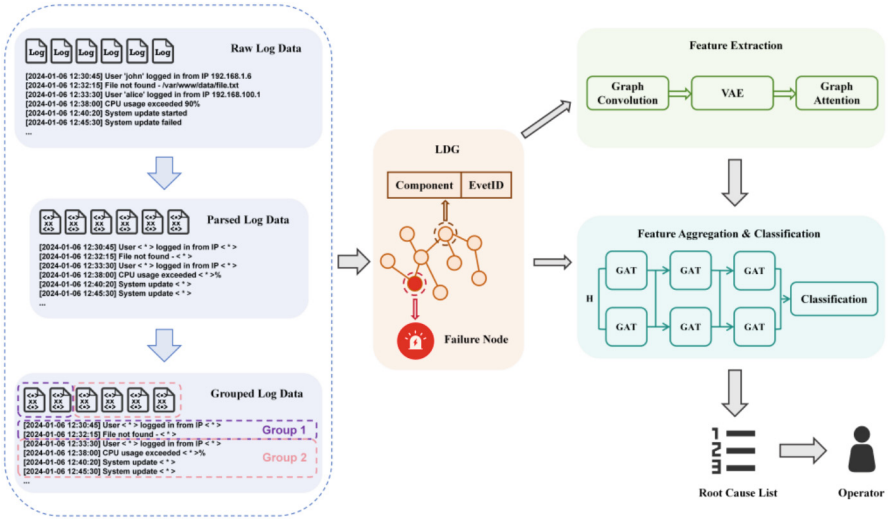

Fig. 2. LGRCL model framework

First, in order to convert the text information in the log data into a structured event representation for subsequent processing, we use Drain to parse the original log data. The parsed results include the type and details of the log event, related component information, etc. Based on the parsed log data, in order to adapt to the problem scenario of root cause location, the parsed logs are grouped by analyzing and combining the

time correlation of different log events, text similarity measurement and the context of the abnormal log, each group is a collection of all logs involved in an event process. This grouping method aggregates log events with similar execution context to form a log group. For each log group, an undirected LDG is constructed, where each node represents an EventID and its corresponding component. In order to assign a value to each node, we first preprocess the text information of each log template, including removing non-character words and stop words, splitting compound words into separate words, etc. Secondly, the pre-trained word embedding model Glove is used to generate vector representations for each word in each log template. In order to take the importance of words into account, Term frequency-inverse document frequency (TF-IDF) is used to measure the weight of each word and build the embedding vector of each log template. The vector representation of each node is its embedding vector [15]. In addition, we add additional binary features to represent fault information, adding a binary value to each node to indicate whether the node is a fault node. After concatenating the binary features to the embedding vector, the value of each node is calculated. When there is a dependency relationship between two nodes (for example, the occurrence of log A depends on the status or output of log B, or the occurrence of log A triggers log B), an edge is added.

Through this construction process, we obtained a LDG for each group of logs, as shown in Fig. 3. The node vectors in the graph contain rich semantic information, which helps to better understand the relationships and dependencies in the log data, providing more accurate features for subsequent root cause location.

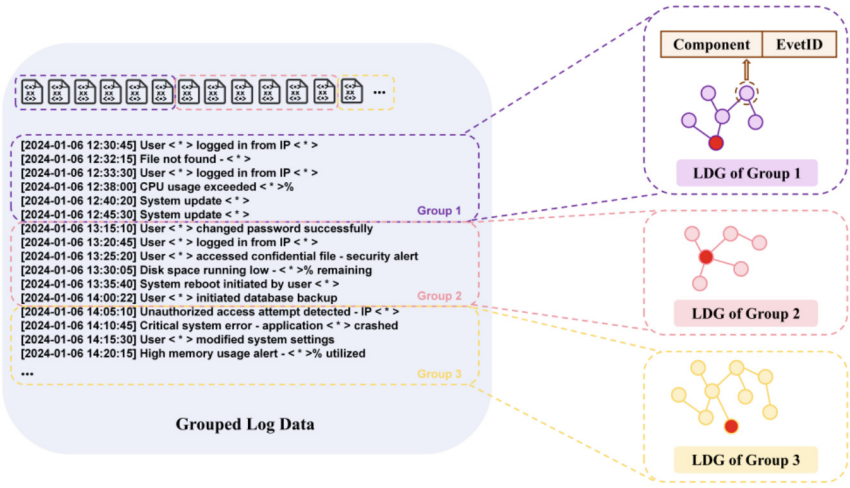

Fig. 3. LDG construction process

3.2 Feature Extraction

By introducing VAE and graph attention in the feature extraction module, the potential features of log data can be better learned, and the generated node features can be focused

on key parts of the nodes, improving the model's ability to capture and understand the correlations in the LDG, thereby improving the accuracy of root cause location. The structure diagram of the feature extraction module VAEGA is shown in Fig. 4.

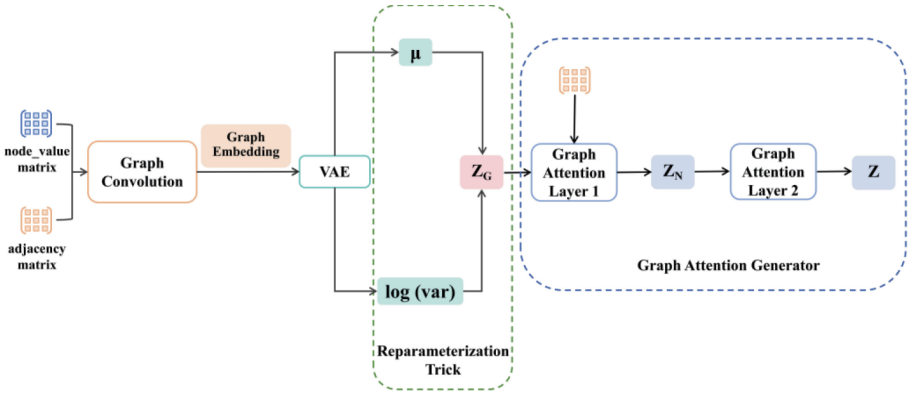

Fig. 4. Feature extraction module structure

We first use a graph convolutional layer to convert the LDG into a higher-level abstract feature representation. Then the VAE layer is introduced to learn the latent characteristics of the data. In order to introduce randomness, we use reparameterization techniques to transform latent variables into random variables with specific distributions. Specifically, we use the mean and log variance to generate a standard normal distributed random noise. This step enables the VAE layer to not only adapt to the distribution of training data during the learning process, but also generate graph features with a certain degree of randomness. The formula for generating graph-level latent features Z_G through the VAE layer is as follows:

$$Z_G = \mu + \epsilon \times \sqrt{\exp(log(var))} \tag{1}$$

Among them, μ and $log(var)$ correspond to the mean and variance of the latent space respectively. In order to sample the latent space, the re-parameterization technique is used to sample $\epsilon \sim \mathcal{N}(0, 1)$ from the standard normal distribution to generate the latent feature Z_G.

Subsequently, in order to more accurately capture the node relationships and features in the LDG, we introduced the stacking of graph attention layers to map graph-level features Z to node-level features. The formula for generating node features Z through the graph attention layer is as follows:

$$e = LeakyReLU\,(a \odot tanh(W \cdot h)) \tag{2}$$

$$Z = Matmul\Big(\big[Softmax(e)\big]^T, h\Big) \tag{3}$$

Among them, Matmul represents matrix multiplication. LeakyReLU means a linear rectifier unit with leakage, which introduces nonlinearity to enhance the expression

ability of the model. a represents the parameter of the attention mechanism, which is a learnable weight. W represents the attention parameter, which is obtained through learning. h represents the feature obtained by linear transformation of node embedding.

Finally, the reconstruction loss is calculated using the squared Euclidean distance. In graph generation tasks, it is very important to preserve the relative relationships and local patterns between nodes. For example, in the problem scenario of network fault root cause location, important system characteristics may be reflected in the local structure of the graph, and the squared Euclidean distance helps the model pay more attention to these important local differences. In addition, for node features containing multiple dimensions, when calculating the reconstruction loss, each dimension is squared and summed to fully consider the similarity of multi-dimensional features. The formula for calculating the reconstruction loss is as follows:

$$Loss = \frac{1}{n}\sum_{i=1}^{n} \|node\_value\_matrix - Z\_matrix\|_2^2 \tag{4}$$

Among them, n represents the feature dimension of each node. *node_value_matrix* represents the node value matrix of the input LDG. *Z_matrix* Represents the node feature matrix generated by graph attention.

3.3 Feature Aggregation and Classification

After feature extraction of nodes in the LDG, feature aggregation is performed on the extracted features. Since fault propagation is a multi-hop behavior, multiple GATs are superimposed one by one to establish a multi-hop fault propagation model. In addition, using multi-head GAT for feature aggregation allows the model to more comprehensively consider different relationships and substructures, thereby improving the overall feature expression ability. It can effectively handle data with diverse and dynamic characteristics such as log data.

Finally, a two-layer dense neural network is used to calculate a score for the aggregated features of each node, and the scores are sorted in descending order. A list of root causes of the fault is obtained and fed back to the engineers so that they can take measures.

4 Experiment and Result Analysis

4.1 Datasets and Environment

The experiment uses three data sets Hadoop [16], Spirit and Thunderbird. Among them, Hadoop [16] is collected from a Hadoop cluster consisting of 46 cores on 5 machines and contains 683 EventIDs. Spirit and Thunderbird contain system logs collected from the BlueGene/L (BGL), Spirit and Thunderbird supercomputing systems located at Sandia National Laboratories respectively. Spirit contains 834 EventIDs and Thunderbird contains 1013 EventIDs. All three datasets contain ground truth labels as well as identifiers used to divide log messages into different groups. If someone want to generalize to other datasets, the dataset must have log message that can extract log templates, and identifiers that can divide logs into groups according to different needs.

4.2 Evaluation Metrics

In order to evaluate the performance of the algorithm, three evaluation indicators are introduced: $AC@k$, $Avg@k$ and MAR. $AC@k$ represents the probability that the first k results given by each algorithm contain the real root causes of all given failure cases. When k is small, the higher the $AC@k$ value, the more accurate the algorithm is in identifying the actual root causes. Due to the reduction of the search space, the efficiency of further search by the operator is improved. For the fault condition of set A, the calculation formula of $AC@k$ is as follows:

$$AC@k = \frac{1}{|A|} \sum_{a \in A} \frac{\sum_{i<k} R^a[i] \in V_{rc}^a}{min\left(k, \left|V_{rc}^a\right|\right)} \tag{5}$$

Where $R^a[i]$ is the ranking result of all metrics of failure condition a. V_{rc}^a is the root cause set of fault condition a. $Avg@k$ evaluates the overall performance of each algorithm $AC@k$ by calculating the average, which is defined as:

$$Avg@k = \frac{1}{k} \sum_{1 \le j \le k} AC@j \tag{6}$$

The average rating (MAR) is the average of the recommendation rankings for all root causes for each failure. It indicates on average how many recommendations should be checked to diagnose a failure, the smaller the better.

4.3 Analysis of Results

Comparison of Root Cause Location Results. Several root cause positioning models related to the LGRCL model are selected for comparison, namely: DejaVu [17], JSS'20 [12], and iSQUAD [18]. The performance comparison of different models is shown in Table 1, and the bold data indicates the optimal value.

As can be seen from Table 1, the $AC@1$ of the LGRCL model on the three data sets are 79.53%, 75.48% and 82.90% respectively. $AC@5$ can reach up to 94.71%. Compared with the DejaVu model, the model has improved by 7.76%, 1.93% and 3.20% respectively. The root cause positioning accuracy is 35.36% higher than the baseline model on average, and the root cause positioning efficiency is 13.71% higher than the baseline model on average. This shows that the construction of LDG and feature extraction method can improve the processing effect of log data and the accuracy of root cause location. Since the DejaVu model converts log data into time series data for feature extraction, the effect is slightly different compared to feature extraction for graph structures. Compared with JSS'20 and iSQUAD, both models have higher improvements. The performance of JSS'20 and iSQUAD is poor first because they do not perform deeper feature learning and representation learning on the input data. At the same time, the methods they use are unsupervised, and systems with large amounts of data are very likely to be interfered by noise and other factors.

Comparison of the Results of Feature Extraction in Different Data Forms. When extracting features from log data, the LGRCL model converts the log data into a LDG to extract features of the graph structure. We adopted another way of processing data for

Table 1. Examples of Correspondence Between EventID, Log Template and Log Message

Dataset Model		LGRCL	DejaVu	JSS'20	iSQUAD
Hadoop	*AC@1*	**79.53%**	71.77%	24.97%	25.49%
	AC@2	**86.28%**	83.61%	32.63%	33.65%
	AC@5	**93.42%**	91.20%	47.26%	58.26%
	Avg@2	**82.91%**	77.69%	28.80%	29.57%
	MAR	**1.60**	1.95	28.06	12.57
Spirit	*AC@1*	**75.48%**	73.55%	22.36%	13.71%
	AC@2	**85.62%**	80.21%	27.04%	17.95%
	AC@5	**89.80%**	86.27%	32.30%	24.80%
	Avg@2	**80.55%**	76.88%	24.70%	15.83%
	MAR	**4.32**	4.86	29.35	33.04
Thunderbird	*AC@1*	**82.90%**	79.70%	38.69%	34.89%
	AC@2	**87.62%**	83.05%	52.46%	59.73%
	AC@5	**94.71%**	89.23%	69.42%	78.50%
	Avg@2	**85.26%**	81.38%	45.58%	47.31%
	MAR	**1.27**	2.60	20.66	12.06

feature extraction, converting log data into time series data for direct feature extraction, and compared the impact of the two data feature extraction methods on the final root cause location results. For the time series feature extraction method, we selected three models: CNN, RNN-AE, and GRU-VAE. The experimental results are shown in Fig. 5.

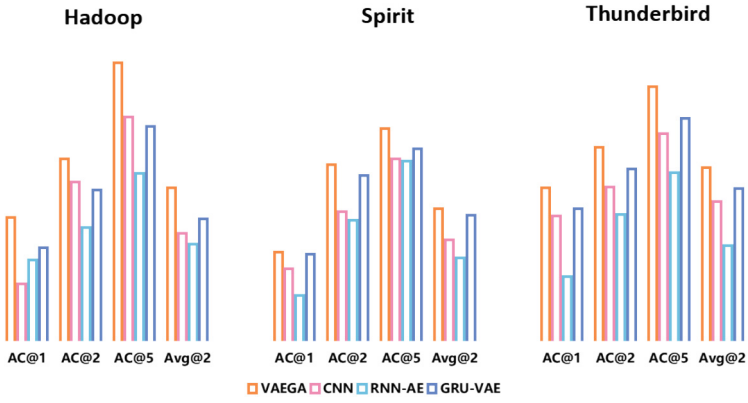

Fig. 5. Comparison chart of feature extraction and root cause location using different data forms

It can be seen from the results that the root cause location results of converting log data into LDG are better than converting them into time series. Since complex dependencies often exist in log data, the interaction and information transfer between nodes are very important. Transforming log data into graph data can naturally represent these dependencies, allowing the model to better capture the complex correlations between nodes and thus more accurately locate root causes. Furthermore, the structure of graph data allows for flexible representation of how different nodes are connected. Compared with time series, graph data is more suitable for expressing dynamic and non-linear relationships. In log data, firings between events can be random, and graph structures can flexibly accommodate this dynamicity.

Ablation Experiment. In order to verify the contribution of different parts of the feature extraction module to the root cause location results, we designed three variants, namely: LGRCL w/o VG, LGRCL w/o V, and LGRCL w/o G. LGRCL w/o VG means that VAE and graph attention are removed in the feature extraction module, and only graph convolution is used for feature extraction. The process of generating node features is as follows:

$$Z' = \sigma\left(\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}} \cdot XW\right) \tag{7}$$

Among them, $\widehat{D}^{-\frac{1}{2}}\widehat{A}\widehat{D}^{-\frac{1}{2}}$ is the normalized adjacency matrix, $\widehat{D}$ is the diagonal matrix of the degree matrix, $\widehat{A}$ is the symmetric normalized adjacency matrix, X is the node value matrix, W is the initialization weight matrix, σ is the activation function (usually ReLU).

LGRCL w/o V means that the VAE part is removed, the preliminary feature representation of the node is generated through graph convolution, and the graph attention adjusts it. The process is as follows:

$$h_i^{(1)} = \sigma\left(\sum\nolimits_{l \in N(i)} W^{(1)} \cdot Z_j^{(0)} + b^{(1)}\right) \tag{8}$$

$$a_{ij} = \frac{\exp(e_{ij})}{\sum_{k \in N(i) \cup \{i\}} \exp(e_{ik})} \tag{9}$$

$$Z_i^{(2)} = \sigma\left(\sum\nolimits_{j \in N(i) \cup \{i\}} a_{ij} \cdot W^{(2)} \cdot h_j^{(1)}\right) \tag{10}$$

Among them, N(i) represents the set of neighbor nodes of node i, a_{ij} is the attention weight processed by softmax, $W^{(2)}$ is the weight matrix of the attention mechanism, $h_i^{(1)}$ is the representation of the node i after the first layer of convolution, and $W^{(1)}$ is the weight matrix of the first layer of convolution. $Z_j^{(0)}$ is the input feature of the node j, and $b^{(1)}$ is the bias term of the first layer of convolution.

The LGRCL w/o G representation removes the graph attention part, produces a preliminary representation of the node through graph convolution, and VAE captures its latent features. The results of the ablation experiment are shown in Fig. 6.

It can be concluded from the results in the figure that the root cause positioning effects of LGRCL w/o VG and LGRCL w/o V are both lower than LGRCL, which

shows that adding two modules will make the positioning results more accurate. VAE is able to better capture the underlying structure and complex distribution of data, thereby generating more representative features. The attention mechanism allows the model to pay more attention to the importance of different nodes in the graph structure during the node feature extraction process. This enables the model to comprehensively consider global and local information and better capture the complex relationships between nodes. And the attention mechanism can dynamically adjust the contribution weight of different nodes to specific nodes, thereby better adapting to the dynamics and complexity of the graph structure. In scenarios that deal with dynamic changes, the introduction of the attention mechanism helps adapt to changes in data. The attention mechanism can help the model capture long-distance dependencies in the graph structure. Compared with models that only use local information, introducing an attention mechanism allows the model to better capture global dependencies when extracting node features, especially for nodes far away from the target node.

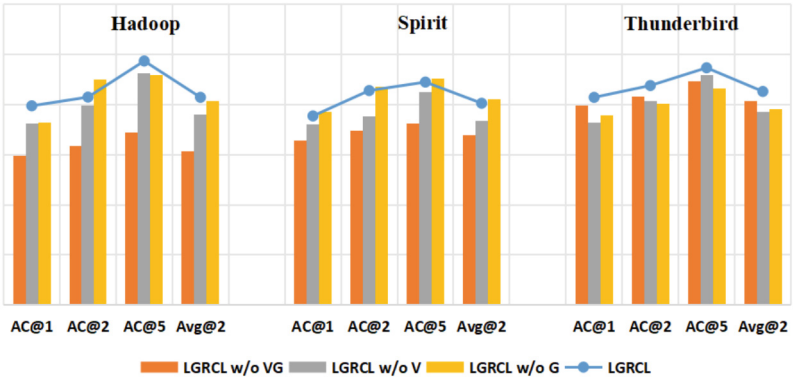

Fig. 6. Comparison chart of ablation experiment results

5 Conclusion and Prospects

In this paper, we propose a root cause localization model based on log data and graph structure, named LGRCL.

(1) The LGRCL model converts unstructured log data into a LDG for root cause location. At the same time, the combination of VAE and graph attention enables the model to extract rich node features in the LDG, and realize the learning and generation of latent features through VAE. This comprehensive design enables our model to more comprehensively capture the correlation relationships of log data and the complex features of the graph structure, improving the accuracy and robustness of root cause location.

(2) The LGRCL model was tested on three data sets. The results showed that the overall root cause location accuracy of the model increased by an average of 35.36%, and the MAR increased by an average of 13.71%, verifying the rationality and effectiveness of the model design.

In future research, we will further consider the dynamics of log data changing over time and design graph structures and feature extraction methods that adapt to dynamic changes to better cope with the root cause location requirements of real-time systems. We can also consider incorporating other information related to log data (such as performance indicators, contextual information) into the model to further improve the effect of root cause location. In addition, we will also explore more data sets and scenarios to verify the versatility of our model in different fields.

Acknowledgments. This study was funded by National Natural Science Foundation of China under grant number U2033205, U2233214.

References

1. Pan, Y., Ma, M., Jiang, X., Wang, P.: Faster, deeper, easier: crowdsourcing diagnosis of microservice kernel failure from user space. In: Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis, pp. 646–657 (2021)
2. He, S., et al.: An empirical study of log analysis at microsoft. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (2022)
3. Zhang, T., Qiu, H., Castellano, G., Rifai, M., Chen, C.S., Pianese, F.: System log parsing: a survey. IEEE Trans. Knowl. Data Eng. (2023)
4. Liu, Y., et al.: UniParser: a unified log parser for heterogeneous log data. In: Proceedings of the ACM Web Conference 2022 (2022)
5. Yan, X., et al.: Aegis: attribution of control plane change impact across layers and components for cloud systems. In: 2023 IEEE/ACM 45th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP 2023) (2023)
6. Ganatra, V., et al.: Detection is better than cure: a cloud incidents perspective. In: Proceedings of the 31st Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE) (2023)
7. Zhang, Y., et al.: CloudRCA: a root cause analysis framework for cloud computing platforms. In: Proceedings of the 30th ACM International Conference on Information & Knowledge Management, pp. 4373–4382 (2021)
8. Wang, H., et al.: Groot: an event-graph-based approach for root cause analysis in industrial settings. In: 2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 419–429 (2021)
9. Wu, L., Tordsson, J., Elmroth, E., Kao, O.: MicroRCA: root cause localization of performance issues in microservices. In: NOMS 2020–2020 IEEE/IFIP Network Operations and Management Symposium, pp. 1–9 (2020)
10. Wu, L., Bogatinovski, J., Nedelkoski, S., Tordsson, J., Kao, O.: Performance diagnosis in cloud microservices using deep learning. In: Hacid, H., et al. (eds.) ICSOC 2020. LNCS, vol. 12632, pp. 85–96. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-76352-7_13
11. Thalheim, J., et al.: Sieve: actionable insights from monitored metrics in distributed systems. In: Proceedings of the 18th ACM/IFIP/USENIX Middleware Conference, pp. 14–27 (2017)
12. Brandón, Á., Solé, M., Huélamo, A., Solans, D., Pérez, M.S., Muntés-Mulero, V.: Graph-based root cause analysis for service-oriented and microservice architectures. J. Syst. Softw. **159**, 110432 (2020)

13. Samir, A., Pahl, C.: DLA: detecting and localizing anomalies in containerized microservice architectures using Markov models. In: 2019 7th International Conference on Future Internet of Things and Cloud (FiCloud), pp. 205–213 (2019)
14. Wu, L., Tordsson, J., Bogatinovski, J., Elmroth, E., Kao, O.: MicroDiag: fine-grained performance diagnosis for microservice systems. In: 2021 IEEE/ACM International Workshop on Cloud Intelligence (CloudIntelligence), pp. 31–36 (2021)
15. Li, Z., Shi, J., van Leeuwen, M.: Graph neural network based log anomaly detection and explanation. arXiv preprint arXiv:2307.00527 (2023)
16. Lin, Q., Zhang, H., Lou, J.-G., Zhang, Y., Chen, X.: Log clustering based problem identification for online service systems. In: Proceedings of the 38th International Conference on Software Engineering Companion, pp. 102–111 (2016)
17. Li, Z., et al.: Actionable and interpretable fault localization for recurring failures in online service systems. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 996–1008 (2022)
18. Ma, M., et al.: Diagnosing root causes of intermittent slow queries in cloud databases. Proc. VLDB Endow. **13**(8), 1176–1189 (2020)