Dynamic Graph Neural Networks-based Alert Link Prediction for Online Service Systems

Yiru Chen*‡§¶, Chenxi Zhang*‡§¶, Zhen Dong†‡§¶, Dingyu Yang‖, Xin Peng‡§¶
Jiayu Ou‖, Hong Yang‡§¶, Zheshun Wu‖, Xiaojun Qu‖, Wei Li‖
‡ School of Computer Science, Fudan University, China
§ Shanghai Key Laboratory of Data Science, Fudan University, China
¶ Shanghai Collaborative Innovation Center of Intelligent Visual Computing, China; ‖ Alibaba Group, China
{yiruchen21, 21212010043}@m.fudan.edu.cn, {cxzhang20, zhendong, pengxin}@fudan.edu.cn
{dingyu.ydy, jiayu.ojy, zheshun.wzs}@alibaba-inc.com, mikong.qxj@alipay.com, jianhao@taobao.com

***Abstract*—A fault in large online service systems often triggers numerous alerts due to the complex business and component dependencies among services, which is known as "alert storm". In a short time, an online service system may generate a huge amount of alert data. This poses a challenge for on-call engineers to identify alerts that are associated with a system failure for root cause analysis. In this paper, we propose DyAlert, a dynamic graph neural networks-based approach for linking alerts that might be triggered by a same fault to reduce the burden of on-call engineers in the fault analysis. Our insight is that alerts are often triggered by alert propagation when a system failure occurs, e.g., alert a would lead to the occurrence of alert b. Whether two alerts should be linked depends on if one alert is triggered by the propagation of the other. Leveraging this insight, we design a dynamic graph (namely *Alert-Metric Dynamic Graph*) that describes the propagation process of alerts. Based on the dynamic graph, we train a neural networks-based model to predict alert links. We evaluate DyAlert with real-world data collected from an online service system running 85 business units and about 30,000 different services in a large enterprise. The results show that DyAlert is effective in predicting alert links and it outperforms the state-of-the-art approaches with an average increase of 0.259 in F1-score.**

***Index Terms*—Linked Alerts, Online Service Systems, Graph Neural Networks**

I. Introduction

Modern online service systems are becoming increasingly large and complex with the development of technologies such as cloud computing and microservices architecture [1]. In such large-scale complex systems, faults (e.g., service interruptions, performance degradation) are inevitable and may lead to economic loss, security risk, and other serious consequences. For example, as the study conducted on 63 data center organizations in the US shows, the average cost of service downtime has steadily increased from $505,502 in 2010 to $740,357 in 2016 [2].

For fast and timely fault detection and diagnosis, observability tools (e.g. distributed tracing, metrics monitoring) and alert management systems are widely used in online service systems. Engineers set many rules in the alert management system, which continuously detects violations of the rules (e.g., transaction success rate below a predefined threshold). If a rule is violated, the alert management system reports an alert and notifies an on-call engineer. Based on the alert content, the on-call engineer confirms whether a fault occurs and handles the fault.

However, in an online service system, a small number of faults may lead to many alerts being reported in a short time, which is called alert storm. This is because faults in complex systems tend to propagate to multiple components, which is known as the cascade effect [3], and a single fault may trigger a number of inter-related alerts. Besides, multiple independent faults may happen within the same period [4], which leads to a mixture of alerts generated by different faults. This poses a huge challenge for on-call engineers to analyze the root cause of a system failure in a short time.

To reduce the burden of on-call engineers in the fault analysis, existing works usually use alert aggregation [4]–[6] or alert linking [7], [8] to identify inter-related alerts. Alert aggregation aims to aggregate alerts caused by the same fault. Alert linking is to determine whether there is a link relation (e.g., duplicate link, sequentially related link [7]) between every two alerts. In addition to their common ability to increase the efficiency of on-call engineers, the latter can also provide links among relevant alerts, which give the on-call engineers more clues and evidence to check and handle the alerts. These alert linking approaches usually use machine learning methods to identify linked alerts based on the semantic information [7], [8] and the system topology information [7] of each alert. Some approaches [8] represent alerts in an alert storm case as an alert sequence to describe the order of occurrence of the alerts.

Our experience shows only using information such as semantics of alert contents and topology of components related to alerts might be insufficient for alert linking. Alert storm occurs often due to alert propagation, i.e., one alert triggers another alert. Whether two alerts should be linked depends on if one alert is triggered by the propagation of the other. Thus, alert propagation information is crucial for accurate alert linking. Unfortunately, it is not considered in existing approaches [7], [8], which may lead to lower accuracy.

In this paper, we propose DyAlert, a dynamic graph neu-

*Yiru Chen and Chenxi Zhang contribute equally to this work.
†Zhen Dong is the corresponding author.

TABLE I
AN EXAMPLE ALERT

Create Time	Severity	Related Metric
2022-07-27 20:05:56	P5	success rate of order creation
Business Unit	**Convergence Count**	**On-call Engineer**
Online Shopping	10	David
Alert Description		
[2022-07-27 20:05 Online Shopping Create Order] The success rate [current value is 89%] in the last two minutes is continuously less than 95%.		

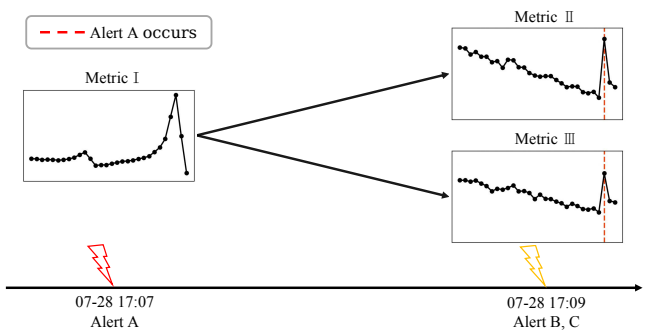


Fig. 1. An example of alert propagation

ral networks-based approach, which takes alert propagation information into account for accurate alert linking. Specifically, DyAlert uses a discrete-time dynamic graph, which is called AMDG, to describe the alert propagation process. Each snapshot in the AMDG is a heterogeneous graph that describes the state of the system when an alert occurs. Based on the dynamic graph, DyAlert uses heterogeneous k-GNNs to learn alert spatial information, and GRU to capture the temporal information of each alert within its active time. After model training, DyAlert predicts the links among alerts according to their spatio-temporal representations. To evaluate the effectiveness and efficiency of DyAlert we conduct a series of experimental studies on an industrial dataset. The results show that DyAlert outperforms existing approaches by 41.8% and 10.1% on average in terms of precision and recall respectively. Moreover, ablation studies further validate the effectiveness of each component in DyAlert. Also, DyAlert is efficient in both training and testing.

In summary, our major contributions are as follows:

- We propose DyAlert, a dynamic graph neural networks-based approach for alert link prediction, which models the alert propagation process as a discrete-time dynamic graph and learns alert spatio-temporal representations from it.
- We conduct experimental studies with real-world alerts from the alert management system in a large enterprise. The results demonstrate the effectiveness and efficiency of DyAlert and confirm the contribution of the components in DyAlert.

The remainder of this paper is organized as follows. We first introduce the background and motivation of our work in Section II. Then in Section III, we describe the detailed design of our proposed approach DyAlert. Section IV shows the experimental studies we conducted and demonstrates the effectiveness of DyAlert. In Section V, we present the existing related approaches. Finally, the conclusion and future extension are shown in Section VI.

II. BACKGROUND AND MOTIVATION

a) Background: Alert management is a process of configuring alert rules, detecting rule violations, diagnosing alerts, and mitigating faults. It usually consists of the following three steps:

Alert Rule Management. The process of engineers setting up alert rules in an alert system (e.g. Prometheus [9]) and managing them is called alert rule management. An alert rule is usually configured for the monitoring metrics and contains a set of conditions that are specific to that monitoring metrics. For example, an alert rule for success rates is defined by a condition that the success rate has dropped by more than 50% in the last 10 minutes. Engineers usually configure different alert rules for different metrics based on their expert experience.

Alert Reporting. The process of the alert system detecting whether a metric has violated the conditions in an alert rule and raising an alert is called alert reporting. An alert usually contains a set of fields that describe the alert. As shown in Table I, the alert contains seven fields, as follows: "Create Time" is the occurrence time of this alert; "Severity" specifies the urgency degree of an alert to be handled; "Related Metric" records the metrics checked by the rule that triggers this alert; "Business Unit" is the team that develops, implements and manages the service or component; "Convergence Count" refers to the number of times other duplicated alerts within a fixed interval are merged into this alert; "On-call Engineer" is the person responsible for handling this alert; "Alert description" is a text describing the alert, including the create time, the corresponding metric, the component, etc.

Alert Handling. The process of the on-call engineers and developers checking, diagnosing, and mitigating faults based on the reported alerts is called alert handling. Since there may be multiple alerts within the same period, on-call engineers need to first link or aggregate alerts manually based on their experiences, the related metrics, and alert descriptions to divide the alerts involved in different faults and assign them to different teams. The developers in the team then locate the root causes of faults based on the alert contents and take mitigation action to eliminate faults. Finally, to guide the future diagnosis improvement of the same type of faults, the developers usually write diagnosis reports for the typical faults after mitigation.

b) Motivation: Figure 1 shows an example of alert propagation in the alert management system of Alibaba. It shows the values of three metrics over time. For the sake of simplicity, we represent the three metrics with Metric I, II and III. Metric

TABLE II
THE RELATED ALERTS OF THE EXAMPLE IN FIGURE 1

No.	Timestamp	Alert Description
A	2022/7/28 17:07	[07-28 17:07 Delivery Service] The success count [current value is 7800] in the last minute was down 27% minute-on-minute. tenant: T application: U
B	2022/7/28 17:09	[07-28 17:09 Order Payment_Peak-hour at Noon] The success count [current value is 110] in the last minute was down 22% minute-on-minute. tenant: T application: V
C	2022/7/28 17:09	[07-28 17:09 Order Home Payment_Split Network Traffic] The success count of N was down more than 13% [peak-hour 1]. The success count [current value is 5600] in the last minute was down 20%>8% minute-on-minute. tenant: T application: V

I, II and III are three business metrics, which are represented as the success count of different business functions. At 17:07 and 17:09, the three Alerts A, B and C generated by the alert rules based on Metric I, II and III occur consecutively because of a spike in traffic in July 2022. The details of these alerts such as alert descriptions are shown in Table II. The 30 metric data points before the alert reporting are also shown in the graph. According to the definition of alert linking [7], there are sequentially related links between Alert A and B and between Alert A and C. There is also a duplicate link between Alert B and C.

Existing alert linking approaches [7], [8] use semantic information of the alert description, and represent alerts as a sequence of alert descriptions [8]. These approaches cannot well identify linked alerts due to the following two reasons:

- The semantic information of the alert descriptions of the linked alerts is not always similar. As the examples shown in Table II, the alert descriptions for Alerts A and B are relatively similar, but Alert C has only a small number of words in its alarm description that are similar to Alerts A and B. Moreover, all the similar words are general words in the alert descriptions, such as "the success count", "was down", and "minute-on-minute". Therefore, alert links may fail to be identified by the common semantic information of the alert descriptions. However, the fluctuations of Metric II and III are almost the same. The link between Alert B and Alert C can therefore be identified by the correlation of the two metrics.
- An alert may cause multiple alerts to occur simultaneously. As the example shown in Figure 1, Alert A leads to the simultaneous occurrence of Alerts B and C. This is because most online service systems monitor metrics sampled at intervals of 30 seconds or 1 minute, which makes it challenging to obtain the accurate order of alerts that occur during the sampling interval. If the alerts are represented as a sequence, it will result in the simultaneous occurrence of Alerts B and C interspersed in an arbitrary manner. However, we can identify the propagation relationships between alerts by analyzing the metrics. As shown in Figure 1, the fluctuation of Metric II and Metric III around 17:09 and Metric I around 17:07 are very similar, which shows a 2 minutes delay in the propagation of the alerts.

Based on the above analysis, we can find that in order to accurately identify linked alerts, it is necessary to consider the semantics of alert, metrics, and alert propagation process. Therefore, we use a discrete-time dynamic graph to describe the alert propagation process and combine the metrics and semantics of alerts.

III. APPROACH

The objective of DyAlert is to automatically and accurately identify linked alerts in online service systems. It takes alerts and metrics as input and trains a dynamic graph neural networks-based model to identify linked alerts.

As shown in Figure 2, the whole process of DyAlert includes four steps. *Alert Embedding* generates a semantic vector representation for each alert by incorporating the alert description and alert severity. *Metric Embedding* generates a vector representation for each metric in the system. *Dynamic Graph Construction* constructs a discrete-time dynamic graph to describe the alert propagation process in the system. *Model Training* trains a dynamic graph neural networks-based alert link prediction model by learning the spatio-temporal information for each alert.

For alert link prediction, DyAlert takes metrics and new alerts as input and predicts whether a new alert is linked to another alert. Following the same process, it generates alert vectors and metric vectors. The model then updates the discrete-time dynamic graph by constructing a new snapshot and feeds the graph into the trained model to produce the result.

A. Alert Embedding

We concatenate the alert description and severity into a sentence and generate a vector representation for it. We only use these two fields because the alert description already contains information about the rest of the fields in the alert content, such as the create time, the related metric, and the corresponding alert rule, etc.

We first tokenize the alert descriptions. Following previous works in incident management [5], [6], we split the alert descriptions into individual words by common separators (e.g., ":", "-"). Moreover, we convert all English words to lowercase. Then we remove all non-verbal symbols (e.g., IP address, time, metric value) and stop words (e.g., "the", "is") from them, but retain the thresholds of their alert rules. For example, the alert description "The success rate [current value is 89%] in the last two minutes is continuously less than 95%" can be tokenized into "success", "rate", "current", "value", "last",

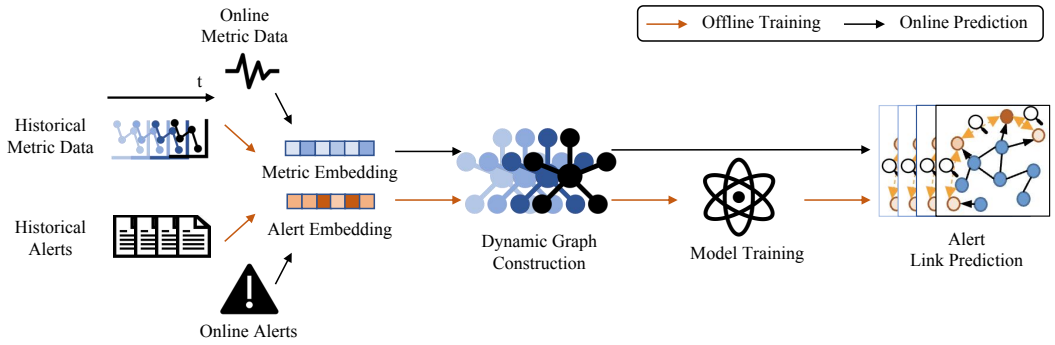


Fig. 2. Overview of DyAlert

"two", "minutes", "continuously", "less", "95%". Finally, we insert the severity of the alert (e.g., P3) at the beginning of the token sequence.

Different from the existing incident management works [7], [8] that employ static word embeddings generated by approaches like CBOW [10] and FastText [11], we use a pre-trained BERT model to generate a vector representation based on the obtained token sequence. BERT is a transformer-based deep learning technique for Natural Language Processing (NLP) pre-training developed by Google [12]. Specifically, we use the pre-trained BERT model provided by [13]. The model's architecture is consistent with the original BERT [12], and its output is a 768-dimensional vector.

Thus for each alert, we can obtain a 768-dimensional vector as the semantic representation of the alert's description and severity.

B. Metric Embedding

The metrics in the system reflect the current state of the system in different dimensions. We transform the recent values of each metric in the system at the create time of an alert into a vector.

We first perform L_2 normalization on the raw values in the time window $[t - w_{metric}, t]$ of each metric, where t is the create time of the alert and w_{metric} is the length of the time window. The reason for normalization is that the value domains of the different metrics in the system are quite different. For example, the success rate takes values between 0 and 1, and the number of requests takes values across the entire range of natural numbers. We then transform the normalized metric values, which are sorted by time, into a $\frac{w_{metric}}{d}$-dimensional vector where d is the sampling interval of metrics.

C. Dynamic Graph Construction

The propagation of alerts in online service systems is a complex process. We design Alert-Metric Dynamic Graph (AMDG) to represent the propagation of alerts in the online service system. AMDG is a discrete-time dynamic graph $G = \{G_1, G_2, ..., G_N\}$ which consists of a series of snapshots at different time intervals. Each AMDG snapshot is a heterogeneous graph G_t, which contains the alerts, metrics and their relations at a particular time. A relation in a AMDG snapshot can be one of the following two types.

- **Cause**: A cause relation from a metric to an alert represents that the metric is the related metric of the alert.
- **Correlation**: A correlation relation represents a statistical correlation between two metrics (e.g., Pearson correlation).

When a new alert occurs, we update the last AMDG snapshot to get a new AMDG snapshot by the following three steps.

Step 1: Addition of new alerts. We add this new alert and all new alerts occurring in the time period after its create time as new nodes to the AMDG snapshot. This is because multiple alerts may occur in the online service system in a short time, and the metrics are usually sampled at intervals of 30 seconds or 1 minute. Generating multiple snapshots in this situation would introduce redundancy in the data and affect the validity of link predictions, therefore we only update the snapshot once in a short time (1 minute in our work). Then, we add a cause relation between each new alert and the corresponding related metric.

Step 2: Removal of expired alerts. We remove alerts that have expired at this time from the AMDG snapshot. The propagation of alerts in online systems has delay effects, therefore they may still have an impact on the system after they have occurred for some time. We define the alert active time t_a to reflect this phenomenon. If the difference between the current time and the alert occurrence time is greater than t_a, the alert is regarded as an expired alert. In particular, for alerts that converge multiple times, we use the time of the last convergence as the create time of that alert.

Step 3: Updating of relations among metrics. We update the relations among all metrics based on the degree of linear correlation between them. Specifically, we calculate the Pearson correlation coefficients based on the raw values in the time window $[t - w_{metric}, t]$ for each pair of metrics. Based on our observation, the historical alerts tend to be linked when the absolute coefficients of related metrics are relatively high. Therefore, we define a threshold α to filter out relations

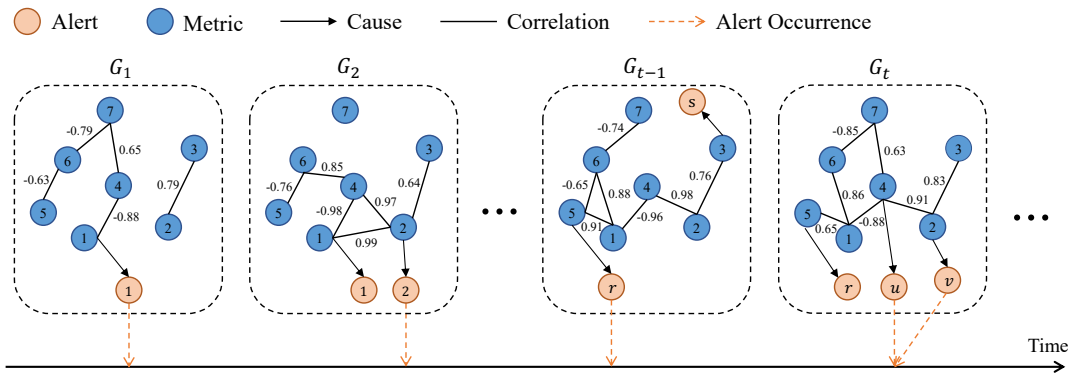


Fig. 3. An example of Alert-Metric Dynamic Graph (AMDG)

with low correlation coefficients. When the absolute value of the Pearson correlation coefficient between two metrics is greater than α, we add the correlation relation between these two metrics and use the value as the weight of this edge. In particular, for metrics with a standard deviation of 0 at this time, we consider that they have no relation with other metrics. This is because a metric with a standard deviation of 0 means that its value has not changed during this time, indicating that it has not been affected by any faults.

For the first snapshot in AMDG, we add all alerts that do not expire at the create time of the snapshot. And we add the cause relations among metrics using the same method as in step 1.

Figure 3 shows an example of AMDG. The orange nodes and blue nodes in each snapshot represent alerts and metrics respectively. The directed edge from a blue node to an orange node represents a cause relation, and the undirected edge from a blue node to the other blue node represents a correlation relation. When a new alert u occurs, the $t-1$-th snapshot is updated to get a new snapshot G_t for the current propagation of alerts. The new alert u and the other new alert v occurring in the time interval are added to the snapshot G_t. Then, the related metrics 4 and 2 of alerts u and v are linked with them respectively by two cause relations. For alert s on the last snapshot G_{t-1}, the current time exceeds its active period, thus it is removed from snapshot G_t. This means it is hardly likely to affect the system and there is no need to link it with other alerts. The other alert r on the snapshot G_{t-1} has not expired at the current time, so it still remains on the snapshot G_t. Besides, there have been some new values of metrics collected until the create time of alert u. The correlation relations on snapshot G_t are updated based on these new metric values. Since the absolute value of Pearson correlation coefficient between metric 4 and metric 7 is greater than α (e.g., α=0.6 in this example), the edge between these two metrics is added on the snapshot G_t.

D. Model Training

We model the alert link prediction task as a link prediction problem in a discrete-time dynamic graph, which aims to train a model that can predict the existence of an edge between two alerts in a discrete-time dynamic graph. As shown in Figure 4, we train a dynamic graph neural networks-based model to learn the spatio-temporal representation for each alert and predict whether it is linked with other alerts.

1) Alert Spatial Representation: For each AMDG snapshot, we use graph neural networks (GNNs) to learn the spatial representations of the alerts. A spatial representation of an alert that combines alerts information, metrics information, and structure information of this AMDG snapshot. The key characteristic of GNNs is to use pairwise message passing to iteratively update the nodes' representations by exchanging information with their neighboring nodes [14]. Specifically, we use k-dimensional GNNs (k-GNNs) [15] which take higher-order graph structures at multiple scales into account to solve the shortcomings of GNNs as Weisfeiler-Lehman algorithm [16]. k-GNNs is designed for homogeneous graphs, however, each AMDG snapshot is a heterogeneous graph that cannot be well handled by k-GNNs. Therefore we extend k-GNNs to heterogeneous graphs by using different k-GNNs layers for each type of relations to extract features.

DyAlert represents each AMDG snapshot as a heterogeneous graph $G_t = \{V^a, V^m, X^a, X^m, E^{ca}, E^{co}, W\}$, where: V^a and V^m refer to a set of alert nodes and metric nodes respectively; $X^a \in \mathbb{R}^{|V^a| \times d^a}$ and $X^m \in \mathbb{R}^{|V^m| \times d^m}$ are nodes attribute matrix consisting of the raw embeddings of alert nodes and metric nodes, in which d^a and d^m are the dimensions of alert embedding and metric embedding; E^{ca} and E^{co} are adjacency matrix of cause relations and correlation relations; W are the edge weight matrix of correlation relations. In every iteration, k-GNNs layers pass node attributes as messages to the target node and update the target node representations based on different type of adjacency matrixes. The node representations after the l-th iterations are obtained by the following equations:

$$
\begin{aligned}
r_p^{m(l+1)} &= \mathbf{F_1} r_p^{m(l)} + \mathbf{F_2} \sum_{q \in E_p^{co}} w_{q,p} r_q^{m(l)} \\
r_{p'}^{a\,(l+1)} &= \mathbf{F_3} r_{p'}^{a\,(l)} + \mathbf{F_4} \sum_{q' \in E_{p'}^{ca}} r_{q'}^{m(l)}
\end{aligned}
\tag{1}
$$

where $r_p^{m(l)}$ and $r_{p'}^{a\,(l)}$ are the representations of the p-th metric

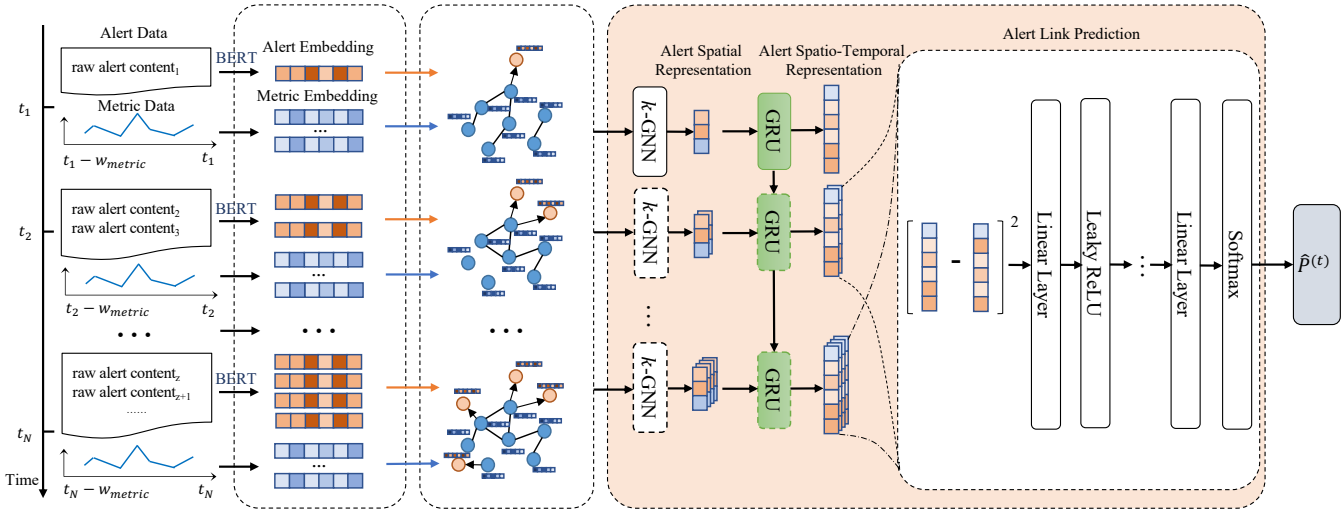


Fig. 4. The architecture of the dynamic graph neural networks-based model in DyAlert

node and the p'-th alert node after l iterations; when $l = 0$, $r_p^{m\,(0)} = X_p^m$ and $r_{p'}^{a\,(0)} = X_{p'}^a$ respectively, which means the original attribute of p-th metric node and p'-th alert node; E_p^{co} and $E_{p'}^{ca}$ are the p-th row of E^{co} and the p'-th row of E^{ca}, which means the sets of source nodes that linked with the p-th metric node and the p'-th alert node respectively according to adjacency matrix; $w_{q,p}$ refers to the edge weight of the correlation relation from the q-th metric node to the p-th metric node in W; $\mathbf{F_i}$, $i \in [1, 4]$ are learnable parameters. Through the above steps, we get the spatial representation of each alert. We use the same model for each AMDG snapshot to obtain the alert spatial representations $R^s = \{R_1^s, R_2^s, \cdots, R_N^s\}$, where $R_t^s = \{r_j^{t\,(L)} | j = 1, 2, \cdots, |V^a|\}$ and $r_j^{t\,(L)}$ is the final representation of j-th alert node in the t-th AMDG snapshot after L iterations.

2) Alert Spatio-Temporal Representation: For a series of AMDG snapshots, we use recurrent neural networks (RNNs) [17] to learn the spatio-temporal representations of the alerts. The spatial representation of an alert is constantly changing due to the changes in the snapshots. For each alert, we treat its spatial representations on different snapshots before it expires as a sequence, thus allowing RNNs to learn the representation of the alert over time, which we call the spatio-temporal representation of the alert. The spatial-temporal representation of an alert combines information about the propagation of alerts and its relations to metrics and other alerts. Specifically, we use Gated Recurrent Unit (GRU) [18] as the RNNs model, which is a gating mechanism in RNNs. In contrast to a Long Short-Term Memory (LSTM), it also has a forget gate but has fewer parameters needed to learn.

For an alert a_j, we feed the alert spatial representation sequence $\{r_j^{y\,(L)}, r_j^{y+1\,(L)}, \cdots, r_j^{y'\,(L)}\}$, where a_j's active period is $[y, y'], (1 \le y \le y' \le N)$, into GRU to get the alert spatio-temporal representation at each AMDG snapshot. The spatio-temporal representation $h_{a_j}^{(t)}$ of the alert a_j after inputting the alert spatial representation $r_j^{t\,(L)}$, $(y \le t \le y')$, at the t-th AMDG snapshot, is defined by the following equation:

$$h_{a_j}^{(t)} = \mathrm{GRU}(r_j^{t\,(L)}, h_{a_j}^{(t-1)}) \tag{2}$$

where $h_{a_j}^{(t-1)}$ is the output spatio-temporal representation of a_j after inputting the alert spatial representation $r_j^{t-1\,(L)}$ at the $t-1$-th snapshot.

3) Alert Link Prediction: After obtaining the alert spatio-temporal representations, we can perform alert link prediction at each AMDG snapshot. We use classification layers to predict whether there is a link between a new alert and the other alert that may be newly added or still existing on the snapshot.

For the link prediction of the new alert a_n and the other alert a_o on the t-th snapshot, the square difference is first calculated with the spatio-temporal representations $h_{a_n}^{(t)}$ and $h_{a_o}^{(t)}$ of them. With square difference, the result is consistent regardless of the order of the two alerts. Then, we use a multilayer perceptron (MLP) with Leaky ReLU activation function to learn the 2-dimensional latent representation Z from the square difference. Finally, the positive probability of the latent representation $\hat{P}_{n,o}^{(t)}$ calculated by a softmax function determines whether two alerts are linked.

$$\begin{aligned} Z &= MLP[(h_{a_n}^{(t)} - h_{a_o}^{(t)})^2] \\ \hat{P}_{n,o}^{(t)} &= \frac{e^{Z_1}}{\sum\limits_{k \in \{0,1\}} e^{Z_k}} \end{aligned} \tag{3}$$

4) Loss Function: At the training phase, we adopt the strategy of moving window with a fixed size w and step Δt on the AMDG to obtain multiple snapshot sequences. We treat each snapshot sequence as a training sample and feed it into the dynamic graph neural networks-based model to predict the links among alerts. Specifically, for each snapshot sequence, we only perform link prediction for the last Δt

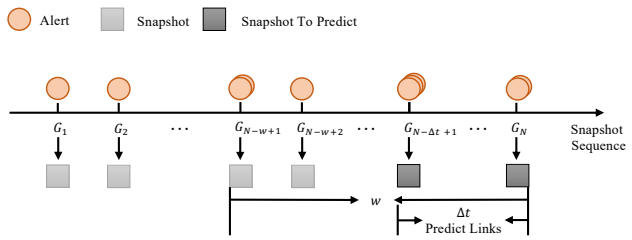


Fig. 5. The process of alert link prediction

snapshots, which is shown in Figure 5. This is because new alerts may be linked to existing alerts, thus we need to learn the spatio-temporal representations of existing alerts. And for each snapshot that needs to be predicted, we only predict whether there is a link between each new alert and the other alert. In this way, we do not have duplicate predictions for the same alert pair.

Moreover, the linked alerts account for only a small proportion, and most alerts are non-linked. The number of linked alert pairs is much smaller than the number of non-linked alert pairs which leads to the class imbalance problem. To alleviate the impact of class imbalance problems, we adopt focal loss [19] as the loss function in DyAlert, which is a dynamically scaled cross-entropy loss function. With the focal loss, we can adjust the hyperparameter modulating factor γ to focus on learning hard misclassified samples and the hyperparameter weighting factor β to reduce the influence from a large number of non-linked samples. The overall loss of each snapshot sequence is calculated by using the following Equation:

$$\mathcal{L} = \sum_{i=w-\Delta t}^{w} \sum_{(n,o)\in \mathbf{Prs^{(i)}}} -[(1-\beta)(1-\hat{P}_{n,o}^{(i)})^{\gamma} {p^*_{n,o}}^{(i)} \log(\hat{P}_{n,o}^{(i)}) + \beta(\hat{P}_{n,o}^{(i)})^{\gamma}(1-{p^*_{n,o}}^{(i)}) \log(1-\hat{P}_{n,o}^{(i)})] \tag{4}$$

where $\mathbf{Prs^{(i)}}$ refers to all alert pairs needed to be predicted in the i-th AMDG snapshot of the snapshot sequence; ${p^*_{n,o}}^{(i)}$ is the ground truth of the pair consisting of alerts a_n and a_o.

Furthermore, we jointly optimize the parameters of heterogeneous k-GNNs, GRU, and MLP instead of training component models separately like [8], thus DyAlert is an end-to-end approach.

E. Online Alert Link Prediction

DyAlert trains a dynamic graph neural networks-based model for alert link prediction. When a new alert occurs, DyAlert conducts alert embedding, metric embedding and AMDG snapshot construction to obtain a new snapshot. If the number of new snapshots does not reach Δt, DyAlert will continue to wait for the next alert. Otherwise, DyAlert predicts linked alerts in new snapshots by feeding the snapshots sequence to the dynamic graph neural networks-based model. Then, the predicted results are sent to engineers immediately for facilitating fault localization and mitigation.

IV. Evaluation

In the evaluation, we conduct a series of experimental studies with real-world data collected from industry to answer the following research questions.

- **RQ1**: How effective is DyAlert on alert link prediction in online service systems?
- **RQ2**: How efficient is DyAlert in model training and online alert link prediction?
- **RQ3**: How do the choices of different hyperparameters affect the performance of DyAlert?

A. Experimental Design

1) Dataset: We use as our dataset alert data generated from July 1st, 2022 to August 1st, 2022 on an online service system that runs 30,000 different services maintained by 85 business units of a large company, Alibaba. In total, we collect 6,995 alerts that are related to 1,310 metrics of the system, e.g., the average number of orders successfully created per minute, response time of a request.

Manually Labelling Alert Links. For a supervised approach, the validity of training data is crucial to its effectiveness. Despite our data being collected from the real-world system, we choose not to use the link labels given by on-call engineers. This is because these links are labeled when handling system emergencies and often are not complete and accurate. To achieve the data with ground truth, we ask two interns in Alibaba to manually analyze if two alerts are inter-related. Given alert a, we only consider alerts that are triggered in the active time of alert a and check if any of them is inter-related to a. If they are inter-related, we make a label between them. Based on our analysis of the data in Alibaba, the delay effect of an alert normally does not last for 10 minutes. Thus, we set the active time of an alert is 10 minutes. With this setting, for the 6,995 alerts, 10,564 pairs of alerts need to be checked if they are inter-related. For labelling, the two interns analyze each alert pair independently and then perform a cross check on their results. For inconsistent results, a third expert with extensive experience is assigned to give an additional assessment to resolve the conflicts by a majority-win strategy. Following this process, they eventually identify 3,255 alert links out of the 10,564 pairs, which are used in the experiments.

Training Set & Testing Set. To evaluate DyAlert, we divide the manually labelled dataset into two distinct subsets: a training set and a testing set. To ensure the model's generalizability and prevent overfitting, we partitioned the dataset based on the occurrence time of alerts. Specifically, $5,836$ alerts that occur from July 1st, 2022 to July 26th, 2022 are used as the training set and $1,159$ alerts that occur from July 27th, 2022 to August 1st, 2022 are used as the testing set.

2) Measures: Since alert link prediction in our work is a binary classification task, we adopt widely-used classification measures for evaluation.

- **Precision** $= \frac{TP}{TP+FP}$ refers to the percentage of relations whose labels are linked in linked relations classified by our model in DyAlert.
- **Recall** $= \frac{TP}{TP+FN}$ is the percentage of linked relations classified by our model in DyAlert out of relations whose labels are linked.
- **F1-score** $= \frac{2 \cdot precision \cdot recall}{precision+recall}$ is the harmonic mean of precision and recall.

On the other hand, alert link prediction aims to link alerts with propagation relations caused by a fault so as to reduce the number of alerts that need to be investigated for diagnosing a fault. Therefore, we use a clustering comparison measure to evaluate the effectiveness of our approach.

- **Adjusted Mutual Information (AMI)** [20] is a widely-used clustering comparison measure based on Shannon information theory to account for chance, represented as $AMI(\mathbf{U}, \mathbf{V}) = \frac{I(\mathbf{U},\mathbf{V})-E\{I(\mathbf{U},\mathbf{V})\}}{\frac{1}{2}(H(\mathbf{U})+H(\mathbf{V}))-E\{I(\mathbf{U},\mathbf{V})\}}$, where $\mathbf{U}$ is the set of ground truth partition, $\mathbf{V}$ is the set of prediction partition, $H(\cdot)$ is the entropy, $E(\cdot)$ is the expected value, and $I(\mathbf{U}, \mathbf{V})$ is the mutual information between $\mathbf{U}$ and $\mathbf{V}$. It takes a value of 1 when the compared partitions are identical, and 0 when the mutual information between the compared partitions equals its expected value. Since many alerts may have no links with others, the reference clustering based on alert links is unbalanced and there exist small clusters. AMI is more suitable for the task of alert link prediction than other clustering comparison measures [21].

3) Baselines: The following approaches are selected as baselines to compare with DyAlert.

- **FP-Growth** [22] is a prefix-tree-based frequent pattern mining algorithm, which can find a series of frequent item sets in large datasets. It is common in industry to mine the frequent pairs in historical manual links between alert templates parsed by Drain [23] with FP-Growth.
- **LiDAR** [7] is a supervised framework for incident link identification. It trains a deep learning model to learn similar semantic representations for the textual descriptions of historically manual linked incidents and adopts node2vec [24] to encode the structure information on the component dependency network which is constructed based on the involved components of the historically linked incidents. Based on these representations, it calculates the semantic and structural similarity by the cosine distance and then identify links.
- **OAS** [8] is a framework to summarize alerts with the links among them. It trains three models respectively to extract semantics from contextual information and the common behavior patterns from the alert description occurrence series and then combine the two pieces of information to classify the link between alerts. All of these models are trained in a supervised manner with historically linked alerts labeled by engineers.
- **DyAlert-T** is a variant of DyAlert that does not consider metrics that are associate with alerts, i.e., only taking semantics of alert content to train a model and predicate alert link.
- **DyAlert-M** is a variant of DyAlert that does not consider semantics of alert contents and remains the same as DyAlert for other parts.
- **DyAlert-G** is a variant of DyAlert that takes semantics of alert contents and fluctuations of metrics for model training, but does not consider sequential information over alert propagation.

4) Implementations and Settings: We implement DyAlert using Python 3.7.15, PyTorch 1.12.1, and PyTorch Geometric 1.7.2. We have open-sourced our code on GitHub [25].

All the experiments are conducted on a cloud elastic compute service with Intel Xeon Platinum 8163 CPU, 368 GB RAM, and Tesla V100 with 32GB GPU memory. The hyper-parameter settings of DyAlert are as follows: the embedding size of semantic representation is set to 768, the embedding size of metric representation w_{metric} is set to 30, the number of heterogeneous k-GNNs hidden layers is set to 2, the hidden sizes of each layer are set to 128 and 256, the hidden size of the GRU layer is set to 256, the number of MLP layers is set to 3, the hidden sizes of each layer are set to 128, 64, and 2 respectively, the threshold α for filtering correlation relations of low linear correlation is set to 0.6, the β and γ in Equation 4 are set to 0.5 and 1.5 respectively, and the collection window length of metrics w_{metric} is set to 30. At the training stage, we adopt Adam [26] to optimize the parameters of the neural networks-based model of DyAlert with an initial learning rate of 0.0001 and train it iteratively for 50 epochs.

B. Effectiveness of DyAlert (RQ1)

Table III shows the results of DyAlert, variants of DyAlert and baseline approaches under comparison described in Section IV-A, where the first column indicates approaches in the evaluation, the following columns present precision, recall, F1-score, and AMI of experiment results. As shown in the table, DyAlert achieves the best performance in terms of precision, F1-score and AMI except for recall. Specifically, the precision, recall, F1-score and AMI of DyAlert are 0.761, 0.758, 0.759 and 0.794 respectively.

Comparison with the variants of DyAlert. Compared to DyAlert-T and DyAlert-M, DyAlert is better than them by 10.1% and 11.1% in terms of F1-score because it considers both semantics of alert contents and fluctuations of related metrics and constructs heterogeneous graphs with them, which can compensate each other to perform better. Moreover, since DyAlert takes the delay effects of an alert propagation process into account and uses GRU to learn the representation of each alert over time, DyAlert outperforms DyAlert-G by 5.6% in terms of F1-score.

Comparison with baseline approaches. DyAlert is effective in alert link prediction and outperforms existing ap-

TABLE III
EFFECTIVENESS OF DIFFERENT APPROACHES

Approach	Precision	Recall	F1-Score	AMI
FP-Growth	0.694	0.541	0.608	0.738
LiDAR	0.520	0.776	0.622	0.487
OAS	0.449	0.819	0.580	0.617
DyAlert-T	0.659	0.724	0.690	0.768
DyAlert-M	0.668	0.699	0.683	0.715
DyAlert-G	0.741	0.698	0.719	0.705
DyAlert	**0.761**	**0.758**	**0.759**	**0.794**

TABLE IV
TRAINING AND TESTING TIME OF DIFFERENT APPROACHES

Approach	Training Time	Testing Time
FP-Growth	2.43s	0.49s
LiDAR	539.07s	2.12s
OAS	22.9s	2.43s
DyAlert	**858s**	**2.74s**

proaches by 41.8%, 10.1%, 25.9% and 33.1% on average in terms of precision, recall, F1-score and AMI.

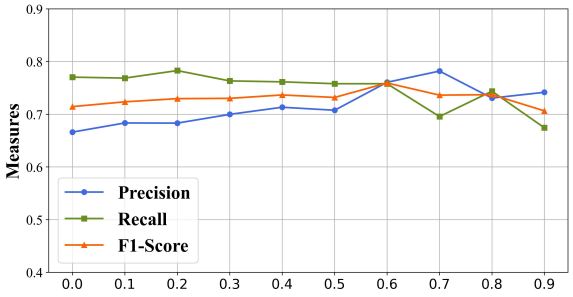
This is primarily due to the following two factors considered in DyAlert. First, except for semantic information of alert contents, recent values of metrics whose anomalies cause alerts are considered in DyAlert. Therefore, DyAlert can identify the underlying links between alerts that may not have similar content but exhibit similar recent fluctuations in their related metrics. Second, DyAlert models alerts' propagation processes with AMDG and learns representations based on AMDG to link alerts. This capability allows DyAlert to identify more challenging alert links with a delay of the propagation between them and achieve higher performance. In contrast, the baseline approaches do not consider such information and may miss the cases above.

Besides, we notice that the results of baseline approaches LiDAR and OAS are different from those reported in their papers. This might be because we use a different dataset. Our dataset is collected from an online service system with extremely complex interactions used in a large company. There are a variety of complicated alert linking scenarios that are not included in their datasets. For example, diverse alert propagation processes at different moments may lead to different link predictions between alerts.

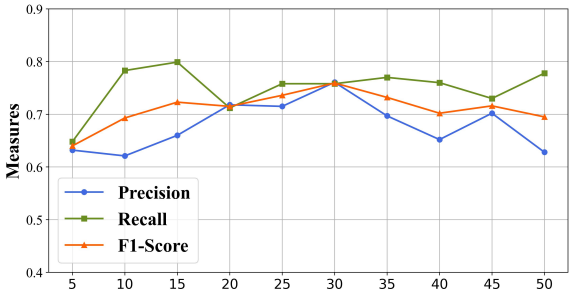
In conclusion, all the components of DyAlert make essential contributions to its effectiveness. Meanwhile, results from real-world data in Alibaba show that DyAlert is effective on alert link prediction, outperforming three existing approaches (i.e., FP-Growth, LiDAR, and OAS).

C. Efficiency of DyAlert (RQ2)

Table IV shows the average training and testing time of DyAlert and the existing approaches. As shown in the table, DyAlert (i.e., 858s) needs more time than FP-Growth (i.e., 2.43s), LiDAR (i.e., 539.07s) and OAS (i.e., 22.9s) during the training phase. This is expected since DyAlert uses a more complex network with more parameters for learning alert propagation based on alert contents and metrics. Considering DyAlert trains the model offline, this time cost is acceptable in practice. In terms of testing time, DyAlert takes 2.74s to predict links between alerts of the testing set. Although DyAlert still has the longest prediction time, which is quite small for performing alert linking for 1,159 alerts. For each AMDG snapshot, DyAlert takes 0.0047s on average to complete alert linking, which is sufficient to assist engineers in the fault diagnosis.



(a) the threshold α for filtering correlation relations of low linear correlation



(b) the collection window length of metrics w_{metric}

Fig. 6. Impact of different hyperparameters

D. Impact of Hyperparameters (RQ3)

DyAlert requires the configuration of two hyperparameters α and w_{metric}. Parameter α is used for filtering the correlation relations among metrics based on Pearson correlation coefficients. Parameter w_{metric} indicates the collection window length of metrics, determining the number of metric values used for calculating Pearson correlation coefficients. We evaluate the impact of these two hyperparameters on the performance of DyAlert.

Figure 6 (a) shows the impact of α on three measures (i.e., precision, recall, and F1-score) of DyAlert. Smaller α usually leads to lower F1-score and precision. When $\alpha = 0$, which means there is no need to filter metric edges, each metric node has an average of 649 neighbors. It is hard for the model in DyAlert to extract valid information and predict alert links based on such complex metric dependencies. On the other hand, bigger α may also cause lower F1-score and recall.

When $\alpha = 0.9$, all edges whose absolute Pearson correlation coefficients are lower than 0.9 are filtered and each metric node only has an average of 3 neighbors. It makes the propagation processes of alerts likely incomplete. According to the results, $\alpha = 0.6$ achieves the best performance in terms of F1-score.

Figure 6 (b) shows how the length of metric collection window w_{metric} impacts the performance of DyAlert. As shown in the figure, both F1-score and precision decrease when w_{metric} gets smaller. It is because smaller w_{metric} results in fewer metric values are used for calculating Pearson correlation coefficient, which is more likely to get an inaccurate coefficient. Besides, bigger w_{metric} also leads to lower F1-score and precision. When w_{metric} is big, the proportion of anomalous fluctuations in all metric values is small, and most metric values are collected before faults occur. This leads to the false alert link prediction for faults that last for a relatively short time. According to the results, we choose $w_{metric} = 30$ for a stable performance. It is worth noting that the hyperparameter selection is highly dependent on the specificity of the data used. The best hyperparameters vary with the data.

E. Threats to Validity and Limitation

1) Threats to Validity: The threats to internal validity mainly lie in the implementation and configurations of baseline approaches and noisy labels. For baseline approaches, we directly use source codes provided by OAS and FP-Growth. But for LiDAR, we implement it by ourselves as it has no publicly available implementations. In order to reduce this threat, we follow the paper of LiDAR and carefully assemble components in the same way. Meanwhile, we choose the best configurations of these approaches for alert link prediction on our dataset. As for noisy labels, since the historical alert links are manually labeled by multiple on-call engineers, and they can only make judgements within a limited time to prevent faults from propagating further, it is inevitable that there are some noisy labels. To alleviate this threat, we reassess labels carefully with the assistance of two interns and a third expert as Section IV-A mentioned. Therefore, we are confident that the amount of noise is small (if it exists).

The threat to external validity mainly lies in the generalizability of our approach. In our experimental studies, we only collect alerts for 32 days from Alibaba, which may affect the diversity of our dataset and the generalizability of our approach. However, Alibaba is a world-leading Internet Service Provider (ISP) with around 1 billion users across the world and the alert data is collected from a large online service system that runs 30,000 different services maintained by 85 business units in Alibaba. Therefore, the system is a typical, representative online service system that generates sufficient complex data. The results of DyAlert on real-world data of Alibaba demonstrate that our approach is generalizable enough to other companies and bring them benefits.

2) Limitation: Although DyAlert achieves superior performance, there are some limitations in the current implementation. First, only the metrics related to historical alerts are considered for the construction of each AMDG snapshot in DyAlert, thus it can not handle new metrics that have not appeared in the historical data during the online prediction phase. Second, the number of monitoring metrics is huge in a large online service system. Constructing AMDG with all the related metrics in historical alerts like the current implementation of DyAlert may lead to high complexity of calculating the correlation coefficient between every two metrics and obtaining spatial representations from complex correlation relations among metrics. Third, DyAlert is implemented to only support offline training of the model, so it is not capable to update in a real-time manner.

V. Related Work

In recent years, there have been a number of works devoted to alert management in both academia and industry. To improve the efficiency of engineers in handling alerts, they focus on optimizing the diagnosis and mitigation procedures from different aspects (e.g., incident prediction, incident triage and alert prioritization)

Some of them aim to tackle the problem of incident prediction [27], [28] for predicting the occurrence of an incident based on alerts' features to reserve time for engineers' proactive actions of mitigation. Chen et al. [27] first propose AirAlert, which can predict general incidents based on the number of different kinds of alerts. Zhao et al. [28] propose eWarn, which conducts careful feature engineering and multi-instance learning to reduce the influence of noisy alerts that can not be handled by AirAlert and generate an interpretable report for a prediction result.

Some works focus on incident triage [29], [30], which aims to assign incidents to the team responsible for them. Chen et al. [29] perform an empirical study of incident triage on multiple large-scale online service systems and discuss possible ways to improve the accuracy of incident triage. Regarding incident triage as a continuous process, Chen et al. [30] propose DeepCT, a deep learning-based approach to incrementally learn knowledge from incident discussions and update incident triage results.

Some other works are devoted to alert prioritization [31], [32], whose objective is to recommend severe alerts with higher priority levels to engineers for priority handling. Jiang et al. [31] propose a peer review mechanism to rank the importance of alerts with alert rules. Since the rule-based approach results in missing severe alerts or wasting time on non-severe alerts, Zhao et al. [32] propose AlertRank, an automatic and adaptive approach that first extracts a set of alert textual and temporal features, and then adopts a ranking algorithm to identify severe alerts.

Different from them, our work is devoted to solving hard fault analysis caused by alert storm. There are some works using alert aggregation [4]–[6] for handling alert storm. Lin et al. [5] apply graph-theoretic approaches to cluster alerts based on their semi-structured contents. Zhao et al. [6] detect alert storms adaptively and cluster alerts with both textual and topological structure similarity. Chen et al. [4] extend their

works for handling multiple alert storm cases within the same period. They use metrics with similar abnormal behaviors to mine the anomalies that do not trigger alerts and apply community detection to find the scopes of different incidents at the same period. There are also some works using alert linking [7], [8] to handle alert storm. Chen et al. [7] also use the semantic and the dependency structure information like existing works for alert aggregation, but construct the dependency network with the involved components of historically linked alerts and train its model in a supervised manner. Considering the system topology is updated continuously, Chen et al. [8] link alerts with the common behavior pattern from the alert description occurrence series except for semantic information. However, all of these existing approaches do not consider the complex process of alert propagation, and they just extract static information to represent alerts.

VI. Conclusion

In this paper, we propose DyAlert, a dynamic graph neural networks-based approach for alert link prediction. It uses a discrete-time dynamic graph constructed by alert semantic information and metric information to describe alert propagation. Based on the dynamic graph, we design a neural networks-based model with heterogeneous k-GNNs and GRU which can learn a spatio-temporal representation for each alert and predict whether the alerts are linked to each other. Experimental studies on the real-world data from Alibaba demonstrate the effectiveness and efficiency of DyAlert. It outperforms three existing approaches (i.e., FP-Growth, LiDAR and OAS).

In the future, we will extend DyAlert to improve the implementation to resolve the limitations mentioned in Section IV-E, recommend potential root causes of faults to further assist engineers in fault mitigation, and on the other hand evaluate DyAlert on various online service systems.

References

[1] J. Lewis and M. Fowler, "Microservices a definition of this new architectural term," Mar. 2014. [Online]. Available: https://www.martinfowler.com/articles/microservices.html

[2] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 111–120. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00020

[3] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "Towards intelligent incident management: why we need it and how we make it," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 1487–1497. [Online]. Available: https://doi.org/10.1145/3368089.3417055

[4] Z. Chen, J. Liu, Y. Su, H. Zhang, X. Wen, X. Ling, Y. Yang, and M. R. Lyu, "Graph-based incident aggregation for large-scale online service systems," in *36th IEEE/ACM International Conference on Automated Software Engineering, ASE 2021, Melbourne, Australia, November 15-19, 2021*. IEEE, 2021, pp. 430–442. [Online]. Available: https://doi.org/10.1109/ASE51524.2021.9678746

[5] D. Lin, R. Raghu, V. Ramamurthy, J. Yu, R. Radhakrishnan, and J. Fernandez, "Unveiling clusters of events for alert and incident management in large-scale enterprise it," in *The 20th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD '14, New York, NY, USA - August 24 - 27, 2014*, S. A. Macskassy, C. Perlich, J. Leskovec, W. Wang, and R. Ghani, Eds. ACM, 2014, pp. 1630–1639. [Online]. Available: https://doi.org/10.1145/2623330.2623360

[6] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, Y. Wu, F. Zhou, W. Zhang, K. Sui, and D. Pei, "Understanding and handling alert storm for online service systems," in *ICSE-SEIP 2020: 42nd International Conference on Software Engineering, Software Engineering in Practice, Seoul, South Korea, 27 June - 19 July, 2020*, G. Rothermel and D. Bae, Eds. ACM, 2020, pp. 162–171. [Online]. Available: https://doi.org/10.1145/3377813.3381363

[7] Y. Chen, X. Yang, H. Dong, X. He, H. Zhang, Q. Lin, J. Chen, P. Zhao, Y. Kang, F. Gao, Z. Xu, and D. Zhang, "Identifying linked incidents in large-scale online service systems," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 304–314. [Online]. Available: https://doi.org/10.1145/3368089.3409768

[8] J. Chen, P. Wang, and W. Wang, "Online summarizing alerts through semantic and behavior information," in *44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022*. ACM, 2022, pp. 1646–1657. [Online]. Available: https://doi.org/10.1145/3510003.3510055

[9] Prometheus.io, "Prometheus," Jan. 2023. [Online]. Available: https://prometheus.io/

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *1st International Conference on Learning Representations, ICLR 2013, Scottsdale, Arizona, USA, May 2-4, 2013, Workshop Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2013. [Online]. Available: http://arxiv.org/abs/1301.3781

[11] T. Mikolov, E. Grave, P. Bojanowski, C. Puhrsch, and A. Joulin, "Advances in pre-training distributed word representations," in *Proceedings of the Eleventh International Conference on Language Resources and Evaluation, LREC 2018, Miyazaki, Japan, May 7-12, 2018*, N. Calzolari, K. Choukri, C. Cieri, T. Declerck, S. Goggi, K. Hasida, H. Isahara, B. Maegaard, J. Mariani, H. Mazo, A. Moreno, J. Odijk, S. Piperidis, and T. Tokunaga, Eds. European Language Resources Association (ELRA), 2018. [Online]. Available: http://www.lrec-conf.org/proceedings/lrec2018/summaries/721.html

[12] J. Devlin, M. Chang, K. Lee, and K. Toutanova, "BERT: pre-training of deep bidirectional transformers for language understanding," in *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, J. Burstein, C. Doran, and T. Solorio, Eds. Association for Computational Linguistics, 2019, pp. 4171–4186. [Online]. Available: https://doi.org/10.18653/v1/n19-1423

[13] Y. Cui, W. Che, T. Liu, B. Qin, S. Wang, and G. Hu, "Revisiting pre-trained models for chinese natural language processing," in *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, ser. Findings of ACL, T. Cohn, Y. He, and Y. Liu, Eds., vol. EMNLP 2020. Association for Computational Linguistics, 2020, pp. 657–668. [Online]. Available: https://doi.org/10.18653/v1/2020.findings-emnlp.58

[14] F. Scarselli, M. Gori, A. C. Tsoi, M. Hagenbuchner, and G. Monfardini, "The graph neural network model," *IEEE Trans. Neural Networks*, vol. 20, no. 1, pp. 61–80, 2009. [Online]. Available: https://doi.org/10.1109/TNN.2008.2005605

[15] C. Morris, M. Ritzert, M. Fey, W. L. Hamilton, J. E. Lenssen, G. Rattan, and M. Grohe, "Weisfeiler and leman go neural: Higher-order graph neural networks," in *The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, The Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, The Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019*. AAAI Press, 2019, pp. 4602–4609. [Online]. Available: https://doi.org/10.1609/aaai.v33i01.33014602

[16] N. Shervashidze, P. Schweitzer, E. J. van Leeuwen, K. Mehlhorn, and K. M. Borgwardt, "Weisfeiler-lehman graph kernels," *J. Mach. Learn. Res.*, vol. 12, pp. 2539–2561, 2011. [Online]. Available: https://dl.acm.org/doi/10.5555/1953048.2078187

[17] W. Zaremba, I. Sutskever, and O. Vinyals, "Recurrent neural network regularization," *CoRR*, vol. abs/1409.2329, 2014. [Online]. Available: http://arxiv.org/abs/1409.2329

[18] K. Cho, B. van Merrienboer, Ç. Gülçehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*, A. Moschitti, B. Pang, and W. Daelemans, Eds. ACL, 2014, pp. 1724–1734. [Online]. Available: https://doi.org/10.3115/v1/d14-1179

[19] T. Lin, P. Goyal, R. B. Girshick, K. He, and P. Dollár, "Focal loss for dense object detection," in *IEEE International Conference on Computer Vision, ICCV 2017, Venice, Italy, October 22-29, 2017*. IEEE Computer Society, 2017, pp. 2999–3007. [Online]. Available: https://doi.org/10.1109/ICCV.2017.324

[20] X. V. Nguyen, J. Epps, and J. Bailey, "Information theoretic measures for clusterings comparison: is a correction for chance necessary?" in *Proceedings of the 26th Annual International Conference on Machine Learning, ICML 2009, Montreal, Quebec, Canada, June 14-18, 2009*, ser. ACM International Conference Proceeding Series, A. P. Danyluk, L. Bottou, and M. L. Littman, Eds., vol. 382. ACM, 2009, pp. 1073–1080. [Online]. Available: https://doi.org/10.1145/1553374.1553511

[21] S. Romano, X. V. Nguyen, J. Bailey, and K. Verspoor, "Adjusting for chance clustering comparison measures," *J. Mach. Learn. Res.*, vol. 17, pp. 134:1–134:32, 2016. [Online]. Available: http://jmlr.org/papers/v17/15-627.html

[22] J. Han, J. Pei, and Y. Yin, "Mining frequent patterns without candidate generation," in *Proceedings of the 2000 ACM SIGMOD International Conference on Management of Data, May 16-18, 2000, Dallas, Texas, USA*, W. Chen, J. F. Naughton, and P. A. Bernstein, Eds. ACM, 2000, pp. 1–12. [Online]. Available: https://doi.org/10.1145/342009.335372

[23] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE International Conference on Web Services, ICWS 2017, Honolulu, HI, USA, June 25-30, 2017*, I. Altintas and S. Chen, Eds. IEEE, 2017, pp. 33–40. [Online]. Available: https://doi.org/10.1109/ICWS.2017.13

[24] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco, CA, USA, August 13-17, 2016*, B. Krishnapuram, M. Shah, A. J. Smola, C. C. Aggarwal, D. Shen, and R. Rastogi, Eds. ACM, 2016, pp. 855–864. [Online]. Available: https://doi.org/10.1145/2939672.2939754

[25] DyAlert, "Dyalert," Aug. 2023. [Online]. Available: https://github.com/FudanSELab/DyAlert

[26] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," in *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Y. Bengio and Y. LeCun, Eds., 2015. [Online]. Available: http://arxiv.org/abs/1412.6980

[27] Y. Chen, X. Yang, Q. Lin, H. Zhang, F. Gao, Z. Xu, Y. Dang, D. Zhang, H. Dong, Y. Xu, H. Li, and Y. Kang, "Outage prediction and diagnosis for cloud service systems," in *The World Wide Web Conference, WWW 2019, San Francisco, CA, USA, May 13-17, 2019*, L. Liu, R. W. White, A. Mantrach, F. Silvestri, J. J. McAuley, R. Baeza-Yates, and L. Zia, Eds. ACM, 2019, pp. 2659–2665. [Online]. Available: https://doi.org/10.1145/3308558.3313501

[28] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *ESEC/FSE '20: 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Virtual Event, USA, November 8-13, 2020*, P. Devanbu, M. B. Cohen, and T. Zimmermann, Eds. ACM, 2020, pp. 315–326. [Online]. Available: https://doi.org/10.1145/3368089.3409672

[29] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice, ICSE (SEIP) 2019, Montreal, QC, Canada, May 25-31, 2019*, H. Sharp and M. Whalen, Eds. IEEE / ACM, 2019, pp. 111–120. [Online]. Available: https://doi.org/10.1109/ICSE-SEIP.2019.00020

[30] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *34th IEEE/ACM International Conference on Automated Software Engineering, ASE 2019, San Diego, CA, USA, November 11-15, 2019*. IEEE, 2019, pp. 364–375. [Online]. Available: https://doi.org/10.1109/ASE.2019.00042

[31] G. Jiang, H. Chen, K. Yoshihira, and A. Saxena, "Ranking the importance of alerts for problem determination in large computer systems," *Clust. Comput.*, vol. 14, no. 3, pp. 213–227, 2011. [Online]. Available: https://doi.org/10.1007/s10586-010-0120-0

[32] N. Zhao, P. Jin, L. Wang, X. Yang, R. Liu, W. Zhang, K. Sui, and D. Pei, "Automatically and adaptively identifying severe alerts for online service systems," in *39th IEEE Conference on Computer Communications, INFOCOM 2020, Toronto, ON, Canada, July 6-9, 2020*. IEEE, 2020, pp. 2420–2429. [Online]. Available: https://doi.org/10.1109/INFOCOM41043.2020.9155219