Alert Summarization for Online Service Systems by Validating Propagation Paths of Faults

JIA CHEN, Fudan University, China
YUANG HE, Fudan University, China
PENG WANG*, Fudan University, China
XIAOLEI CHEN, Fudan University, China
JIE SHI, Fudan University, China
WEI WANG, Fudan University, China

For online service systems, alerts are crucial for root cause analysis as they capture symptoms triggered by system faults. In real-world scenarios, a fault can propagate across multiple system components, generating a large volume of alerts. Various approaches have been proposed to summarize alerts into incidents to accelerate root cause analysis, using the topology information. However, these approaches focus solely on connectivity, neglecting the semantics of the topology, which significantly impacts their performance. In this paper, we introduce ProAlert, a novel topology-based approach that summarizes alerts into incidents by validating fault propagation paths. ProAlert first unsupervisedly learns fault propagation patterns from historical alerts and system topology offline. It then uses these patterns to validate fault paths in real-time alerts, leading to more accurate incident summarization. Moreover, the fault propagation paths provided by ProAlert improve the interpretability of incidents, assisting maintenance engineers in understanding the root causes of faults. To demonstrate the effectiveness and efficiency of ProAlert, we conduct extensive experiments on real-world data. The results show that ProAlert outperforms state-of-the-art approaches.

CCS Concepts: • **Software and its engineering** → **Maintaining software**.

Additional Key Words and Phrases: Alert Summarization, System Maintenance, Online Service System

ACM Reference Format:
Jia Chen, Yuang He, Peng Wang, Xiaolei Chen, Jie Shi, and Wei Wang. 2025. Alert Summarization for Online Service Systems by Validating Propagation Paths of Faults. *Proc. ACM Softw. Eng.* 2, FSE, Article FSE097 (July 2025), 23 pages. https://doi.org/10.1145/3729367

1 Introduction

As the scale and complexity of online service systems continue to increase, faults have become inevitable. These faults can lead to significant economic losses and other severe consequences. For example, during Prime Day in 2019, Amazon faced an estimated loss of up to 100 million dollars due to a one-hour period of service downtime, significantly impacting its largest annual sale event [3].

*Peng Wang is the corresponding author.

Authors' Contact Information: Jia Chen, Fudan University, Shanghai, China, chenj17@fudan.edu.cn; Yuang He, Fudan University, Shanghai, China, yahe23@m.fudan.edu.cn; Peng Wang, Fudan University, Shanghai, China, pengwang5@fudan.edu.cn; Xiaolei Chen, Fudan University, Shanghai, China, chenxl21@m.fudan.edu.cn; Jie Shi, Fudan University, Shanghai, China, jshi22@m.fudan.edu.cn; Wei Wang, Fudan University, Shanghai, China, weiwang1@fudan.edu.cn.

ACM 2994-970X/2025/7-ARTFSE097
https://doi.org/10.1145/3729367

According to research carried out on 63 data center organizations in the United States, it is evident that the average expense incurred due to service downtime has shown a consistent rise [7].

Faults can stem from various factors, encompassing network glitches, hardware problems, software bugs, environmental conditions like temperature and humidity, and even human errors. In order to promptly detect faults and notify maintenance engineers for timely repairs, the typical way is to deploy numerous monitoring mechanisms on both software and hardware. These monitoring mechanisms generate real-time data, such as KPIs [30, 34, 35], logs [10, 13, 15, 18, 22, 31], and traces [20, 25, 26, 36]. When anomaly occurs in the monitored data, corresponding alerts are generated to record the symptom of the underlying fault. Then, maintenance engineers immediately receive a notification to analyze the newly generated alerts and address the fault.

In practice, maintenance engineers often find themselves overwhelmed by a substantial number of alerts, because the fault can propagate along the topological relationships between system components, resulting in multiple alerts [3, 5, 32]. The online service system may suffer from alert storm due to a large number of alerts generated within a short period [32]. The substantial number of alerts can make it difficult for maintenance engineers to manage, potentially resulting in some faults indicated by alerts not being resolved promptly and leading to significant losses.

To prevent maintenance engineers from being overwhelmed by a flood of alerts and enable them to quickly locate faults, the approaches are utilized to summarize the alerts derived from the same fault. In the early stage, maintenance engineers deploy predefined rules based on the personal maintenance experience. These rules cluster alerts of the same fault into a group, called incident. However, manually defining rules is labor-intensive and sub-optimal in performance for the large and complex systems [3]. Next, many approaches are proposed to summarize alerts [3, 5, 7, 17, 32]. These approaches mainly utilize either the alert contents (also called the semantic information) or other information to correlate alerts. OAS [3] connects alerts based on the semantic and co-occurrence information of alerts. DyAlert summarizes KPI-related alerts based on abnormal fluctuations in KPIs [7].

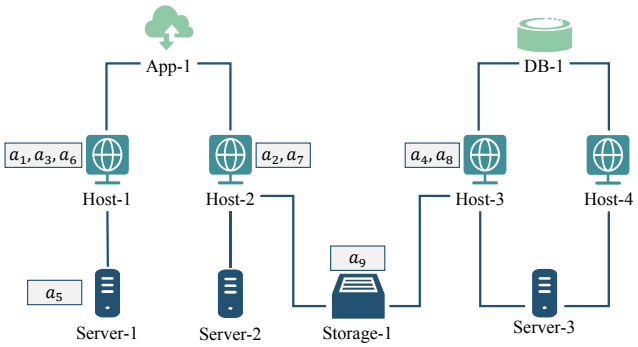


Fig. 1. The hierarchical topology of an online service system.

Recently, as the quality of CMDB has steadily improved [27], more works investigate how to utilize the relationships between alert-affiliated system components, also called as the topological information, to summarize alerts. Fig. 1 shows a small topology which includes 10 components of various types, and the alerts issued from these components are showed in Table 1 and Table 2. The rationale of these approaches is that if two alerts occurring adjacent in time belong to connected components, they should be derived from the same fault and be summarized together.

For example, StormSum [32] determines the relationship between alerts based on the shortest distance of system components. LiDAR [5] first employs GNN models to embed the topology nodes,

and then combines both topological information and semantic information to summarize alerts. Compared to approaches utilizing only the alert contents, these topology-based approaches can achieve higher summarization performance.

However, these topology-based approaches only consider the *connectivity* of the topology, but ignore the *semantic* information of the topology itself, which will cause uncorrelated alerts to be incorrectly connected. We illustrate it based on the system topology in Fig. 1 and the alerts in Table 1 and Table 2 respectively.

Table 1. The First Example: Network Interruption

No.	Timestamp	Component	Content
a_1	2023-01-25 21:55:05	Host-1	The host is unreachable, indicating a potential occurrence of network exception.
a_2	2023-01-25 21:55:06	Host-2	Connection error to host-1 with the app-1-job-1 Job.
a_3	2023-01-25 21:55:12	Host-1	Connection error to host-2 with the app-1-job-1 Job.
a_4	**2023-01-25 21:55:35**	**Host-3**	**User: oracle-1, instance: inst-1, lost connection to host-4.**
a_5	2023-01-25 21:56:35	Server-1	Network disconnection, name: server-1, rack: A50-2H, please contact administrators!

Table 2. The Second Example: Storage Insufficiency

No.	Timestamp	Component	Content
a_6	**2023-02-05 12:54:12**	**Host-1**	**CPU usage exceeds 80%, CPU idle rate is 16.68%.**
a_7	2023-02-05 12:55:05	Host-2	Current disk usage is 80.30%, which exceeds the threshold 80.0%.
a_8	2023-02-05 12:55:06	Host-3	Oracle user: oracle-1, instance: inst-1, table: app_server_table-1, table space usage: 92.5%, exceeds 90.0%.
a_9	2023-02-05 12:55:15	Storage-1	Used capacity exceeds 80%. Please monitor closely!"

Illustrative Example. Fig. 1 shows a small topology. Server-1, server-2, and server-3 are physical machines. Host-1, host-2, host-3, and host-4 are virtual machines. Host-1 and host-2 are deployed on server-1 and server-2, respectively. Both host-3 and host-4 are deployed on server-3. Host-1 and host-2 jointly provide the functionalities of the application service, app-1. Storage-1 is a storage device, providing disk space for host-2 and host-3. Host-3 and host-4 are the components of the database, db-1. Table 1 shows an incident caused by a network interruption fault (except a_4). Table 2 shows an incident caused by a storage insufficiency fault (except a_6).

Fig. 2 illustrates the propagation path of these two incidents. In the first incident, due to the network interruption of server-1 (a_5), Host-1, deployed on it, also reports network abnormality alerts (a_1 and a_3). Host-2, which together with Host-1 forms the application service App-1, also issues the network connection loss alert. So, this incident consists of 4 alerts, a_1, a_2, a_3 and a_5. However, considering Storage-1 and Host-3 are closely connected to Host-2, and a_4 (on Host-3) has similarity content with a_2, previous topology-based approach will wrongly add a_4 into this incident. In fact, a_4 indicates a database instance connection loss, which is uncorrelated to the other alerts.

Similarly, in the second incident in Table 2, alerts a_7, a_8 and a_9 are triggered by a storage insufficient fault. a_6 is a CPU alert triggered by a peak business period and is uncorrelated to these alerts. However, it can also be incorrectly added into this incident by previous approach.

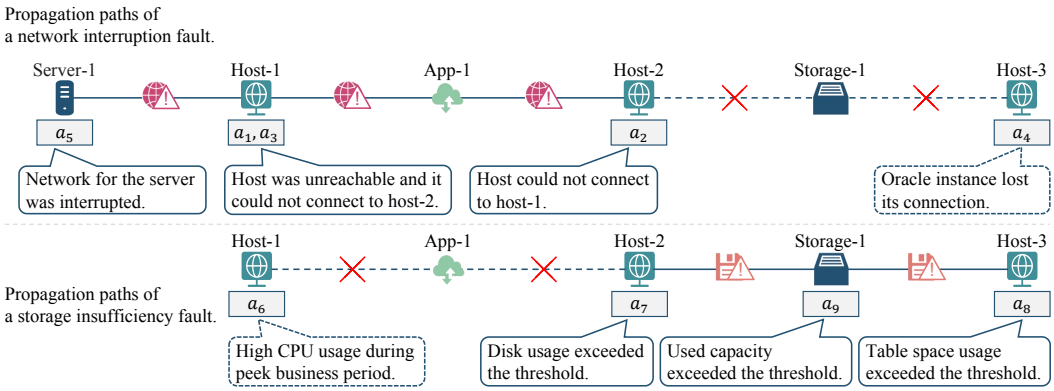


Fig. 2. Propagation paths of two faults.

According to Fig. 2, the propagation paths of the network interruption fault in the first example are: <server-1, host-1>, <host-1, app-1>, and <app-1,host-2>. The propagation paths of the storage insufficiency fault in the second example are: <storage-1, host-2>and <storage-1, host-3>.

Apparently, one edge (or one path) will play different roles when considering different incidents. A network interruption fault of server-1 does not affect the access of Host-2 and Host-3 to the storage device, Storage-1. A storage insufficiency fault of Storage-1 can propagate to all system components that use the device, such as Host-2 and Host-3. As a result, although Host-2 and Host-3 are connected through Storage-1, in the first example, a_2 and a_4 should not belong to the same incident, whereas in the second example, a_7 and a_8 belong to the same incident.

In summary, when summarizing alerts based on the component connectivity, we should consider whether the semantics of the edges between system components align with the semantics of the alerts that propagate through them. However, previous topology-based approaches fail to consider this factor. Thus they may result in inaccurate alert correlations, which is verified by the experiments in Section 4 and case study in Section 5.1. Furthermore, some existing approaches are supervised and require a large number of labeled data [5]. However, maintenance engineers are typically overwhelmed with their workload. It is difficult for maintenance engineers to spare sufficient time for data labeling. As a result, existing approaches exhibit the following issues:

- Existing topology-based approaches do not consider the semantics of propagation path. This issue can result in inaccurate alert correlations.
- Existing approaches require labeled data to learn alert correlations,which are not feasible for a real production environment.

To address these issues, this paper proposes ProAlert, a novel approach to investigate the semantics of the propagation paths, called as *propagation pattern*. In the offline stage, ProAlert uses an unsupervised approach to mine the fault propagation patterns. where the key represents a pair of system component types and the value corresponds to a set of semantic vectors. These vectors are semantic representations that describe faults which frequently propagate between the pair of specific system component types. Then, in the online stage, ProAlert utilizes the mined propagation patterns to validate the propagation paths of faults indicated by newly reported alerts. This process can accurately summarize alerts into incidents. In addition, the mined propagation paths assist maintenance engineers to better understand the faults reflected by incidents.

In summary, the major contributions of this paper are as follows:

- We propose a novel approach to unsupervisedly mine fault propagation patterns according to history alerts and system topology. The approach first transmits alert semantics to the edges between system components, and then mines faults that frequently propagate between specific system component types by clustering semantics on the edges.
- We propose ProAlert, an innovative approach that utilizes the mined patterns to validate the propagation paths of faults indicated by newly reported alerts, thereby accurately summarizing alerts online. Unlike existing approaches, ProAlert further provides incidents with the propagation paths of faults. Thus, the incidents offer higher interpretability and facilitate understanding for maintenance engineers.
- We conduct experiments using real-world data to evaluate our approaches. The results demonstrate the effectiveness and efficiency of our approaches, proving that our approaches perform better in alert summarization compared to existing approaches.

2 Background

In this section, we introduce the topology of an online service system and formally define alerts and alert summarization problem involved in this paper.

2.1 System Topology

All alerts are reported from components of an online service system, which typically has a complex and highly distributed hierarchical topology [19]. Specifically, an online service system consists of multiple layers of components that work together to provide high quality services to a large number of users. A system component is a distinct, modular unit of software or device that performs a specific function or set of functions within a online service system. Each component interacts with other components to contribute to the overall service of the system.

At the lowest layer, physical devices such as server machines, storage, and networking equipment form the underlying infrastructure. These resources usually are virtualized into environments as virtual resources, which can be dynamically created, scaled, and terminated as needed. At the highest layer, engineers can deploy various services with a broad range of functionalities.

For example, in Fig. 1, server-1, server-2, and server-3 are physical machines, while host-1, host-2, host-3, and host-4 are virtual machines. Host-1 is deployed on server-1, and host-2 is deployed on server-2. Both host-3 and host-4 are deployed on server-3. Host-1 and host-2 jointly provide the functionalities of app-1.Storage-1 is a storage device that provides disk space for both host-2 and host-3. Additionally, host-3 and host-4 function as components of the database, db-1.

The topology of an online system, which encompasses the topological relationships between system components, is crucial for system maintenance. Therefore, in a real production environment, system topology is typically carefully maintained by engineers and stored in a dedicated Configuration Management Database (CMDB). In this paper, we define the topology of an online service system as $T = \{O, E, \Delta\}$. O is a vertex set, in which each $o \in O$ is a system component. E is the set of edges that represent topological relationships between system components. Each edge, $e \in E$, connects two components, o_e and o'_e, that is, $e =< o_e, o'_e > (o_e, o'_e \in O)$. Δ is the set of system component types, in which each $\delta \in \Delta$ is the type of a system component, such as "server" and "storage device". For a system component, $o \in O$, its type is denoted as $\delta_o \in \Delta$.

2.2 Preliminary and Problem Statement

An *alert* is a notification reported when monitored data, such as KPIs, logs, and traces, in an online service system is abnormal, recording the symptom of a fault. As shown in Table 1 and Table 2, an alert, denoted as a_i $(1 \leq i \leq n)$, consists of timestamp, t_i, alert content, c_i, and system component, $o_i \in O$. The timestamp, t_i, represents the time when the alert is reported. The alert content, c_i,

describes the symptom of a fault. The system component, o_i, is the component where the alert is reported in an online service system. In addition, based on timestamps, we define reported alerts as an alert sequence, $S = [a_1, a_2, \cdots, a_n]$.

An *incident*, denoted as I, is a group of alerts triggered by the same fault. Given history alerts, S, and system topology, T, the aim of our work is to mine how faults propagate between system components and to summarize newly reported alerts into incidents, $\{I_1, I_2, \cdots, I_N\}$, online. N is the number of result incidents, which is usually much smaller than the number of alerts.

3 Approach

Fig. 3 shows the overview of ProAlert. ProAlert consists of two stages, offline stage and online stage. For input alerts in both stages, *alert embedding* parses the variables in alert contents and generates semantic representations for alerts.

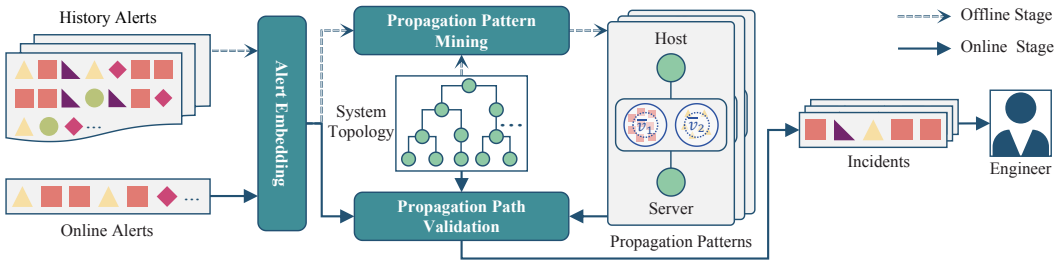


Fig. 3. The overview of ProAlert.

In offline stage, by leveraging system topology and history alerts, *propagation pattern mining* aims to mine the propagation patterns of faults between system components. A pattern is a key-value pair, where the key is a pair of system component types and the value is a set of vectors. The set of vectors are semantic representations describing faults which frequently propagate between the pair of specific system component types. In online stage, *propagation path validation* utilizes the patterns mined in offline stage to validate propagation paths of faults indicated by newly reported alerts, thereby accurately summarizing alerts into incidents.

3.1 Alert Embedding

An alert content comprises two parts: variables and constants. Variables capture system metrics at the time the alert is reported, such as CPU usage and disk availability. Constants describe the symptom caused by a fault. *Alert embedding* aims to accurately represent the semantic information of an alert content. Variables usually do not carry specific semantic information about faults and may interfere with the accuracy of semantic representation. The remaining text, after removing variables from the alert content, is defined as a template. Alerts of the same template represent the same type of fault symptom. Thus, *alert embedding* first parses alert contents into templates that consist solely of constants [14, 37]. In this paper, we adopt Drain [14], which is an established online parser [3, 7, 37], to parse alert contents in both offline and online stages.

Then, each template is embedded into a vector to represent its semantic information. Different from existing works [3, 5, 7] that use static word embeddings generated by approaches like CBOW[23] and FastText [24], we employ a pre-trained text embedding model, BGE [29], to generate vector representations for each template. BGE is based on BERT [9], which is a popular transformer-based deep learning model for Natural Language Processing (NLP) pre-training. As

a result, for each input alert, a_i ($1 \leq i \leq n$), its content, c_i, is parsed into a template and then embedded as a vector-based semantic representation, v_i.

3.2 Offline Stage

In offline stage, we propose *propagation pattern mining* for ProAlert to mine the propagation patterns of faults from history alerts and system topology, consisting of two steps:

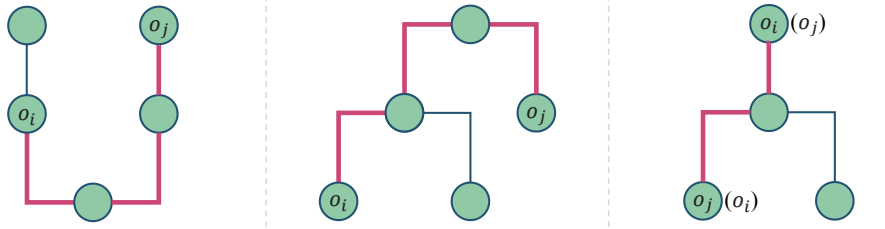
- *Semantics transmission.* This step transmits the semantics of alerts that potentially belonging to the same fault to the edges of paths where the fault might have propagated, based on system topology.
- *Pattern mining*. This step clusters the semantics of the edges to obtain the propagation patterns, $\mathcal{P}$. Each pattern $P \in \mathcal{P}$ is a key-value pair, in which the key is a pair of system component types and the value is a set of vector-based semantic representations. P describes faults which frequently propagate between a pair of system component types.

3.2.1 Semantics Transmission. Alerts of the same fault usually are reported within short time intervals and the system components reporting these correlated alerts are typically close to each other [3, 5, 7, 32]. Therefore, for each alert, a_i, in S ($1 \leq i \leq n$), *semantics transmission* first adopts a time window, $[t_i - w, t_i]$ to find alerts that are reported within the window. Then, for an alert, a_j ($1 \leq j \leq n$), within the window, *semantics transmission* further assesses whether they may be correlated with a_i based on system topology.

Due to the hierarchical topology of an online service system discussed in Section 2.1, lower-layer components serve upper-layer components by providing basic functions, while upper-layer components depend on and orchestrate lower-layer components to deliver advanced services. Accordingly, a fault propagates in a cascading manner through system topology [8]. Thus, for a_j to be correlated with a_i, there should be a finite length between o_i and o_j (not exceeding a maximum value τ) and adheres to one of the following structural constraints:

- o_i and o_j directly or indirectly depend on the same system component in a lower layer.
- o_i and o_j directly or indirectly serve the same system component in a higher layer.
- o_i (o_j) directly or indirectly depends on o_j (o_i).

It should be noted that τ does not represent the length of the longest possible fault propagation chain. Instead, it denotes the distance between two components where consecutive alerts are triggered during fault propagation. There may be intermediate components between them that do not report alerts due to limitations in monitoring mechanisms.



1. o_i and o_j depend on the same system component in a lower layer

2. o_i and o_j serve the same system component in a higher layer

3. o_i (o_j) depends on o_j (o_i)

Fig. 4. The examples of semantics transmission.

Fig. 4 shows three examples of the above structural constraints. By searching system topology, *semantics transmission* can obtain all the paths satisfies the above structural constraints. For all obtained paths, *semantics transmission* transmits the semantic representations, v_i and v_j, to the edges of the paths. Specifically, for an edge e in an obtained path, v_i and v_j are added to a set, R_e. It records the semantic representations transmitted to the edge.

3.2.2 Pattern Mining. After *semantics transmission*, `pattern mining` mines fault propagation patterns, which describe the types of semantic information related to faults that can be propagated between two components. For example, a fault propagated between a storage device and a host is generally related to storage rather than network.

Specifically, `pattern mining` first aggregates semantic representations on edges formed by the same component type pair and then clusters the aggregated semantic representations. After removing outliers, each cluster thus can reveal a fault that frequently propagate between two types of system components.

For a component type pair, $\{\delta_{o_e}, \delta_{o'_e}\}$ ($e =< o_e, o'_e >\in E$), its clustering results are denoted as $V_{\{\delta_{o_e},\delta_{o'_e}\}} = \{\overline{v}_1, \overline{v}_2, \cdots, \overline{v}_z\}$. Each $\overline{v}_k$ ($1 \leq k \leq z$) is the centroid of the semantic representations in a cluster, representing a fault. Thus, the key-value pair, $<\{\delta_{o_e}, \delta_{o'_e}\}, V_{\{\delta_{o_e},\delta_{o'_e}\}}>$, is a pattern revealing faults that frequently propagate between the component types, δ_{o_e} and $\delta_{o'_e}$.

Existing popular clustering approaches contain partitional approaches like K-Means [21], density-based approaches like DBSCAN [11], and hierarchical approaches [12, 28]. In this paper, we choose DBSCAN [11], which is a density-based approach, for ProAlert due to the following reasons. First, ProAlert has no prior knowledge about the number of clusters, while DBSCAN can infer it based on data distribution. Second, DBSCAN can work with most distance measures [28]. Third, DBSCAN can automatically identify outliers in data and discard them. After the clustering process, we obtain a set of propagation patterns, denoted as $\mathcal{P}$, each pattern $P \in \mathcal{P}$ is a key-value pair.

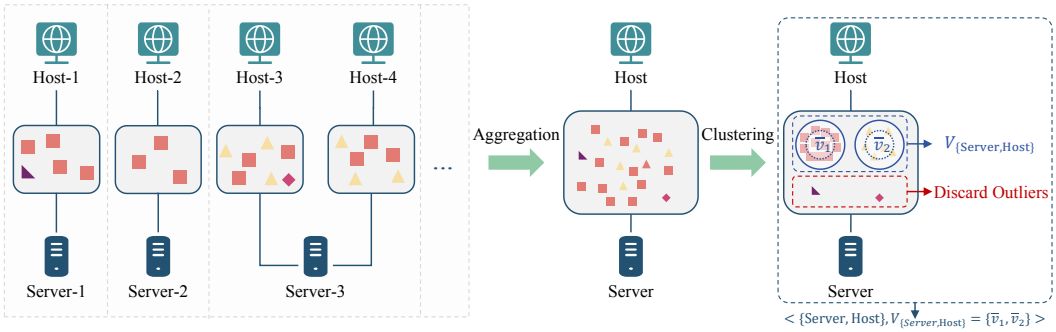


Fig. 5. A toy example of pattern mining process.

Fig. 5 shows a toy example of `pattern mining` process. `Pattern mining` first aggregates semantic representations on edges formed by the component type pair, {Server, Host}. Then, `pattern mining` clusters them based on semantic similarity. The clustering result, $V_{\{\text{Server,Host}\}}$, represent the propagation pattern of faults between servers and hosts. In the pattern, each of $\overline{v}_1$ and $\overline{v}_2$ reveals a fault that frequently propagates between the two types of system components in history. Moreover, outliers that lack sufficient similar semantic representations to form clusters are regarded as mistransmitted and are discarded.

3.3 Online Stage

In online stage, we propose *propagation path validation* for ProAlert to summarize alerts. Propagation Path Validation utilizes mined patterns to find the propagation paths of faults indicated by newly generated alerts. A propagation path is a path between system components associated with alerts triggered by same fault, revealing the propagation process of the fault. If a propagation path exists between the system component of a new alert and that of a previous alert, it indicates that both alerts are triggered by the same fault. Consequently, the new alert is added to the existing incident containing the previous alert. If no such path exists, the new alert is added to a new incident.

Specifically, when a new alert, a_i, is reported, ProAlert first examines the time window, $[t_i - w, t_i]$. If an alert a_j exists within this time window, ProAlert checks whether there is a path, $L = [e_1, e_2, \cdots, e_m]$ ($o'_{e_k} = o_{e_{k+1}}, 1 \leq k \leq m-1$), connecting their system components, where $o_{e_1} = o_i$, $o'_{e_m} = o_j$, and m is the length of the path. This path must meet the structural constraints specified in Section 3.2.1. If these conditions are satisfied, ProAlert then validates whether the path conforms to the mined patterns.

ProAlert first measures the validity of the fault indicated by a_i propagating through the path. For the k-th edge of the path ($1 \leq k \leq m$), ProAlert computes max cosine similarity between v_i and the semantic representations in $V_{\{\delta_{o_{e_k}}, \delta_{o'_{e_k}}\}}$, denoted as s^i_k. In addition, ProAlert adopt the minimum value of s^i_k to represent the validity of the fault indicated by a_i propagating through the path:

$$\hat{s}_i = min_{1 \leq k \leq m}(s^i_k). \tag{1}$$

Similarly, ProAlert measures the validity of the fault indicated by a_j propagating through the path, denoted as $\hat{s}_j$. If a_i and a_j are reported by the same system component, there is no need to measure the validity of the path and both values of $\hat{s}_i$ and $\hat{s}_j$ equal to the maximum value 1. Then, for a_i and a_j, by integrating the validity of the path between their system components with their semantic similarity, ProAlert measures the correlation score between a_i and a_j, as defined below:

$$score_{i,j} = \alpha * cos(v_i, v_j) + (1 - \alpha) * min(\hat{s}_i, \hat{s}_j), \tag{2}$$

wherecos computes cosine similarity, αadjusts ProAlert's feature attention, If $score_{i,j}$ does not exceed the threshold, λ, a_i and a_j are triggered by different faults.

In addition to measuring the correlation score, ProAlert verifies the dependencies of o_i and the system components involved in the previous incident containing a_j. Specifically, ProAlert checks whether these components depend on the same lower-layer component or serve the same upper-layer component, either directly or indirectly. If both are satisfied, L is a propagation path, indicating that a_i and a_j belong to the same fault. Consequently, a_i is appended to the previous incident containing a_j. If multiple alerts within the time window, $[t_i - w, t_i]$, can be correlated with a_i, ProAlert chooses the alert with the maximum correlation score. If no alerts can be correlated with a_i, a new incident is created for a_i.

4 Evaluations

In order to evaluate the effectiveness and efficiency of ProAlert, we conduct a comprehensive study with the purpose of exploring the following research questions:

- RQ1: How does ProAlert perform in alerts summarization?
- RQ2: How accurate are the patterns mined by ProAlert?
- RQ3: How does ProAlert perform in real-time efficiency?

- RQ4: How does the time window width w affect the summarization performance?
- RQ5: How does the path length threshold τ affect the summarization performance?
- RQ6: How does the attention factor α affect the summarization performance?
- RQ7: How does the correlation threshold λ affect the summarizing performance?
- RQ8: How robust is ProAlert to system updates?

4.1 Dataset

The datasets used in this study are from two real online service systems operated by a large commercial company, A, which provides services for over a billion users across hundreds of countries. Specifically, two datasets, S1 and S2, are utilized in this study.

Dataset S1 spans from January 20, 2023, to February 20, 2023, containing 51,352 alerts parsed into 1,003 distinct templates. The service system topology of S1 consists of 126,413 system components, categorized into 14 types, and connected by 190,027 topological relationships. Dataset S2 covers the period from April 1, 2023, to April 30, 2023, and includes 125,830 alerts, which are parsed into 1,257 distinct templates. The service system topology of S2 consists of 65,041 system components, categorized into 22 types, and connected by 140,450 topological relationships.

In the experiments, the first 80% of the alerts in each dataset are used for training, while the remaining 20% are used for testing. Specifically, for S1, the first 41,082 alerts are used for training, and the last 10,270 alerts were used for testing; for S2, the first 100,664 alerts are used for training, and the last 25,166 alerts are used for testing.

4.2 Baselines

In experiments, ProAlert is compared with the following state-of-the-art and baseline approaches:

- LiDAR[5]: LiDAR is a deep learning framework designed to identify correlations between alerts in large-scale online service systems. LiDAR incorporates the semantic information of alerts and structural information extracted from history correlated alerts to identify correlations between new alerts. In experiments, we adopt the same time window mechanism of ProAlert to summarize alerts into incidents according to the correlations mined by LiDAR.
- StormSum[32]: StormSum is an alert summarization approach in AlertStorm [32]. StormSum constructs a similarity matrix that considers both textual and topological similarities between alerts. The textual similarity is calculated by Jaccard distance, and the topological similarity is calculated by the shortest path length between system components where alerts are generated. Then, the approach uses a weighted sum to calculate the overall similarity. Finally, DBSCAN [11] is applied for clustering alerts based on the similarities between them.
- OAS[3]: OAS (Online Alert Summarizing) is a supervised learning approach that leverages three deep learning models to summarize alerts online: ASR (Alert Semantics Representation), ABR (Alert Behavior Representation), and ACT (Alert Correlation). ASR is designed to mine the common semantic information between alerts. ABR focuses on capturing the common behavioral information between alerts. ACT (Alert Correlation) integrates the semantic and behavioral information to determine the correlations between alerts.
- ProAlert-T: This is a variant of ProAlert. It does not consider the propagation paths of faults and relies solely on the semantic information extracted in Section 3.1 to determine the correlation between input alerts.
- ProAlert-M: This variant of ProAlert considers both semantic information and the propagation paths of faults to determine the correlation between input alerts. However, it does not incorporate the structural constraints proposed in Section 3.2.1, while the other components remain consistent with the original ProAlert.

4.3 Manual Labelling

Manually identifying all possible correlations between alerts is labor-intensive due to the high alert volume and interleaved faults. However, determining if an incident contains uncorrelated alerts is more efficient. Thus, we adopt the following strategy for data labeling.

For test data, two experts are invited to independently label the results of each experimental approach, with a third expert resolving conflicts by voting. An incident is considered incorrect if it contains uncorrelated alerts or invalid propagation paths between the involved system components. However, if an incident contains only a subset of alerts triggered by a fault and has no invalid propagation paths, it is considered correct, as it still accurately summarizes correlated alerts. To facilitate expert analysis, system topology is stored in a Neo4j database. Moreover, incidents containing only a single alert do not contribute to alert summarization but also do not contain uncorrelated alerts. Therefore, they are by default considered correct and are not submitted for expert labeling. Additionally, experts do not label the same incident multiple times.

For training data, we use the following process to label alert incidents.

(1) Employ our unsupervised approach, ProAlert, to summarize alerts into incidents.
(2) Experts assess the correctness of each incident.
(3) Experts are asked to decompose each incorrect incident into multiple correct ones.
(4) Merge overlapping incidents until no further merges are possible.
(5) Experts assesses whether two incidents indicated the same fault if the last alert of one and the first alert of the other occurred within an hour. If so, they are merged. This step continues until no further merges are needed.

The above process ensures accurate and complete training data labeling while reducing expert workload. All manual labeling steps involves two independent experts, with conflicts resolved by a third expert via majority vote. Finally, for S1, 861 unique incidents are labeled in the training data, and 329 unique incidents in the test data; for S2, 4937 unique incidents are labeled in the training data, and 929 unique incidents in the test data.

4.4 Metrics

In experiments, we adopt two metrics, the summarization accuracy (SA) and the valid compression ratio (VCR) [3], to evaluate the effectiveness of experimental approaches. In addition, for each approach, we record the time cost of summarizing alerts and the average number of alerts processed per second to evaluate the efficiency of the approach. Specifically, the summarization accuracy is defined as $SA = \frac{n_c}{n}$, where n_c is the number of alerts in correct incidents. The valid compression ratio is defined as $VCR = (1 - \frac{N_c + n_w}{n}) * 100\%$, where n is the total number of input alerts, N_c is the number of correct incidents, and n_w is number of alerts in incorrect incidents.

The summarization accuracy indicates the accuracy of summarization results by measuring the proportion of correctly summarized alerts to the total number of alerts. The valid compression ratio represents the summarization ability of an experimental approach, which ignores the contribution of incorrect incidents to alert summarization. The higher the summarization ability of an experimental approach, the higher the valid compression rate and summarization accuracy.

4.5 Settings

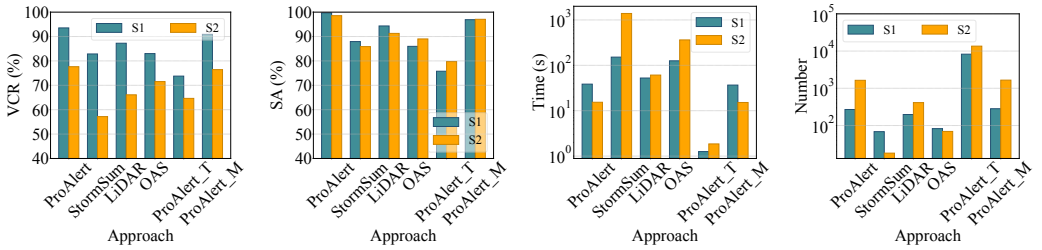
We implement our approaches using Python 3.10.13 and PyTorch 2.0.1. Experiments are conducted on a server with an Intel(R) Xeon(R) Gold 6330 CPU, 500GB RAM, and a 4090Ti GPU with 24 GB memory. For the dataset used in experiments, we employ the first 80% of the data as the training set, and the remaining 20% as the test set.

The default parameter configuration is as follows: the time window width, w, is set to 15 minutes, the path length threshold, τ, is set to 4, the attention factor, α, is set to 0.7, and the similarity threshold, λ, is set to 0.88. To ensure a fair performance comparison, all the parameters of comparison approaches are adjusted to achieve the maximum valid compression ratio.

4.6 Results

We evaluate experimental approaches from the following aspects to answer the proposed research questions.

4.6.1 RQ1: The performance of ProAlert in alert summarization. Fig. 6 illustrates a comparative analysis of the alert summarization performance between ProAlert, its two variants, and other baseline approaches. Fig. 6(a) details the valid compression ratio (VCR) of experimental results, whereas Fig. 6(b) presents the summarization accuracy (SA) of experimental results. The experimental results demonstrate that ProAlert consistently outperforms other approaches on both datasets in terms of the VCR metric, while maintaining high SA. Specifically, it achieves a VCR of 93.53% and an SA of 99.71% on S1, and a VCR of 77.63% and an SA of 98.55% on S2.



(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 6. The performance of different alert summarization approaches.

Comparison with variant approaches. Based on experimental results, we can observe that ProAlert demonstrates superior performance compared to ProAlert-T in terms of VCR and SA. This suggests that the propagation path mechanism effectively enhances the performance of alert summarization. Furthermore, ProAlert also demonstrates better performance than ProAlert-M, highlighting the benefits of incorporating structural constraints in the propagation path mechanism for alert summarization.

Comparison with baseline approaches. As S1 and S2 correspond to different business systems, the compressibility of alerts varies between S1 and S2. We can find that both StormSum and LiDAR exhibits lower VCR and SA compared to ProAlert. Although both LiDAR and StormSum take into account the topological relationships between system components to correlate alerts, they can only identify whether there is a relatively close relationship between alert-affiliated components. They cannot confirm whether a valid fault propagation path exists between these components. Therefore, they may summarize uncorrelated alerts into the same incident.

Furthermore, the VCR and SA of OAS are also lower than ProAlert. Because, while OAS uses the co-occurrence behavior of alerts for correlation, it also fails to establish whether a valid fault propagation path exists between the system components.

Efficiency of experimental approaches. Fig. 6(c) shows the time cost of alert summarization, while Fig. 6(d) shows the average number of alerts processed per second during summarization

for each approach. According to the results, we can find that the efficiency of ProAlert adapts well to online alert summarization. LiDAR and OAS depend on deep learning models to mine correlations between alerts, which involve a large amount of computation. StormSum requires DBSCAN to correlate alerts online over a period. ProAlert, however, only adopts DBSCAN to learn the propagation patterns of faults between system components offline.

In addition, ProAlert_T has the minimal time cost, and ProAlert_M has the similar time cost to the original ProAlert. This indicates that in ProAlert, validating the propagation paths of faults indicated by alerts takes the majority of the time cost. Although S2 contains more alerts than S1, its corresponding system topology is simpler with fewer topological relationships, as described in Section 4.1. Since the majority of the time in ProAlert is spent on validating propagation paths, the processing time for ProAlert and ProAlert_M on S2 is actually lower than on S1.

4.6.2 RQ2: The accuracy of patterns. We invite experts to directly evaluate the accuracy of the patterns mined by ProAlert. For each pattern, experts review the clustered alert templates. A pattern is considered incorrect if any template reveals anomalies that do not propagate through the topological relationships between the component types specified by the pattern. Two experts independently evaluates the correctness of each pattern. Conflicts are resolved by a third expert through majority vote. Table 3 shows the experimental results. Accuracy is the proportion of correct patterns. We can find that ProAlert mines 138 patterns from dataset S1 and 550 patterns from S2. Both datasets achieve over 88% pattern accuracy, demonstrating the effectiveness of our approach.

Table 3. Propagation Pattern Results

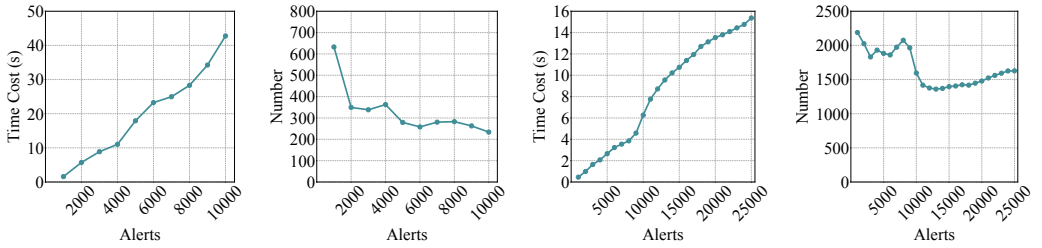
Dataset	Total Patterns	Correct Patterns	Wrong Patterns	Accuracy
S1	138	122	16	88.41%
S2	550	492	58	89.45%

It should be noted that incorrect patterns are not entirely incapable of contributing to alert summarization. For an incorrect pattern, while irrelevant alert templates in the clustering results may cause uncorrelated alerts to be summarized into the same incident, valid alert templates with correct semantic relationships in the clustering results may still accurately summarize alerts.

4.6.3 RQ3: The real-time efficiency of ProAlert. To evaluate the real-time efficiency of ProAlert, we measure the time cost and the average number of alerts processed per second by ProAlert for every 1,000 alerts, as shown in Fig.7. Accoring to Fig. 7(a) and Fig. 7(c), the time cost of ProAlert grows approximately linearly for both datasets, demonstrating stable efficiency across varying alert volumes. For both datasets, as shown in Fig. 7(b) and Fig. 7(d), although there are some fluctuations, the average number of alerts processed per second is higher at the beginning compared to later stages. This is attributed to the relatively fewer alerts in the initial time window, which leads to less time spent on validating propagation paths.

Additionally, for S1, ProAlert processes over 200 alerts per second on average, and for S2, it processes over 1280 alerts per second, meeting the online efficiency requirements of both systems. Although S2 has more alerts than S1, S2 features a simpler topology with significantly fewer topological relationships, as described in Section 4.1. Therefore, the efficiency of ProAlert efficiency in S2 is higher than in S1, as most of the time consumption is spent on validating propagation paths.

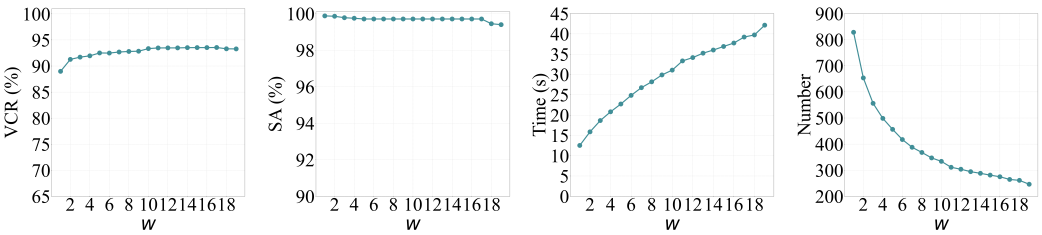
4.6.4 RQ4: Impact of time window width w. To evaluate the impact of the time window width, we use S1 to assess the performance of ProAlert in alert summarization for different values of w.



(a) Time cost on S1. (b) Average number of alerts processed per second on S1. (c) Time cost on S2. (d) Average number of alerts processed per second on S2.

Fig. 7. Experimental results of real-time efficiency.

Fig. 8 shows the experimental results. According to Fig. 8(a), when the time window width, w, becomes larger, the performance of ProAlert improves at first. However, when w exceeds about 15 minutes, the VCR metric gradually stabilize. According to Fig. 8(b), the SA metric remains stable at approximately 99% under different time window widths. Fig. 8(c) and Fig. 8(d) illustrate that as the time window, w, becomes larger, the time cost of alert summarization increases and the number of alerts processed per second decreases.



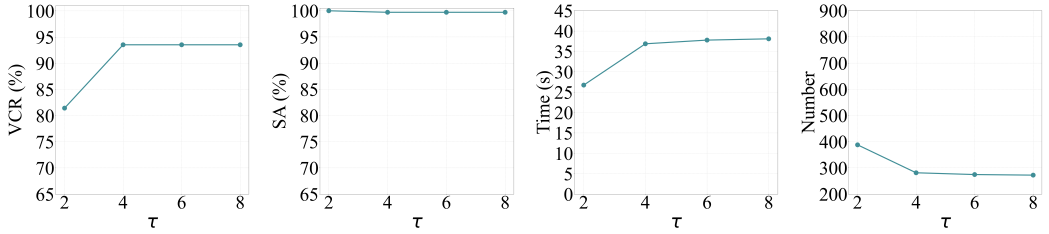
(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 8. Experimental results of varying window width w.

According to the results, we can find that while a larger window allows ProAlert to correlate more alerts, in a real production environment, once the window size is sufficient for ProAlert to correlate alerts of the same faults, further increasing the window size only diminishes the efficiency of ProAlert. In addition, while the window width, w, can impact the number of alert correlations identified by ProAlert, its impact on the accuracy of these identified correlations is minimal. Therefore, time window width, w, should be selected to ensure it is sufficiently large for ProAlert to correlate alerts of the same faults, while avoiding further enlargement that would compromise the efficiency of ProAlert.

4.6.5 RQ5: Impact of path length threshold τ. The impact of the path length threshold, τ, on the performance of ProAlert using S1 is presented in Fig.9. According to Fig. 9(a), increasing the threshold, τ, results in higher VCR. However, enlarging the threshold beyond 4 does not yield additional improvements in summarization performance. This is because, in the experimental service system, the propagation path length of an fault between two system components generally does not exceed 4. According to Fig. 9(b), the SA metric remains stable at approximately 99% under different path

length thresholds, which demonstrates that the threshold, τ, has minimal impact on the accuracy of identified alert correlations.

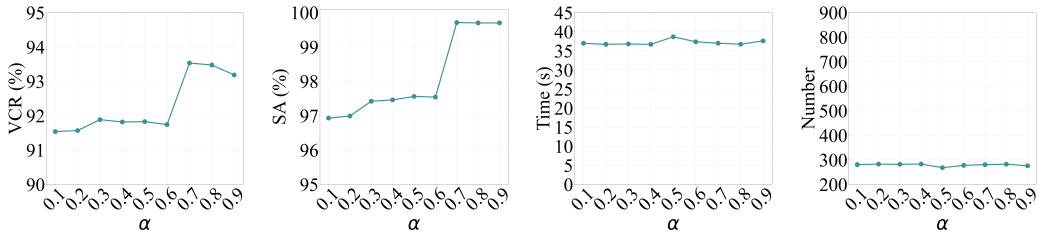


(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 9. Experimental results of varying path length threshold τ.

Moreover, Fig.9(c) and Fig.9(d) demonstrate that as τ increases, the time cost of alert summarization also increases. This is because increasing τ results in a higher computational load for ProAlert. Therefore, the threshold should be selected to the minimum value that ensures optimal summarization effectiveness while maintaining summarization efficiency.

4.6.6 RQ6: Impact of attention factor α. We also adopt S1 to evaluate the impact of attention factor α. Fig.10 illustrates the impact of the attention factor, α, on the performance of ProAlert. The factor, α, adjusts the weights of features when computing the correlation scores between alerts. As shown in Fig. 10(a) and Fig. 10(b), setting the attention factor, α, to either an extremely low value (0.1) or a high value (0.9) cannot enable ProALert to achieve optimal performance. ProAlert has the best performance when α is set to 0.7.

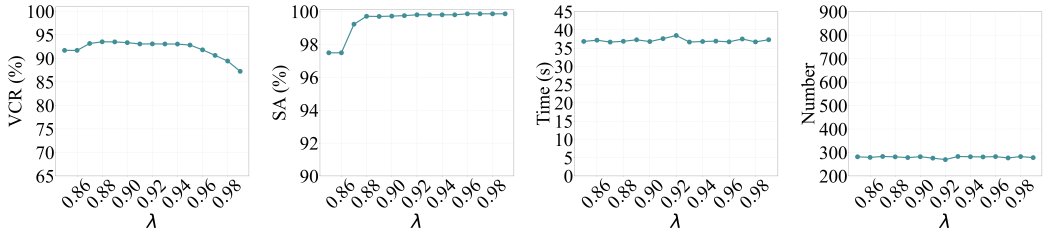


(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 10. Experimental results of varying attention factor α.

Therefore, both textual similarity and the propagation path validity play crucial roles in our approach, and satisfactory summarization performance can only be achieved by considering both types of features. In addition, Fig.s 10(c) and Fig.s10(d) indicate that α has minimal impact on the efficiency of ProAlert.

4.6.7 RQ7: Impact of correlation threshold λ. Fig.11 illustrates the impact of correlation threshold λ on ProAlert's performance with S1. According to Fig. 11(a), as the threshold, λ, increases, the



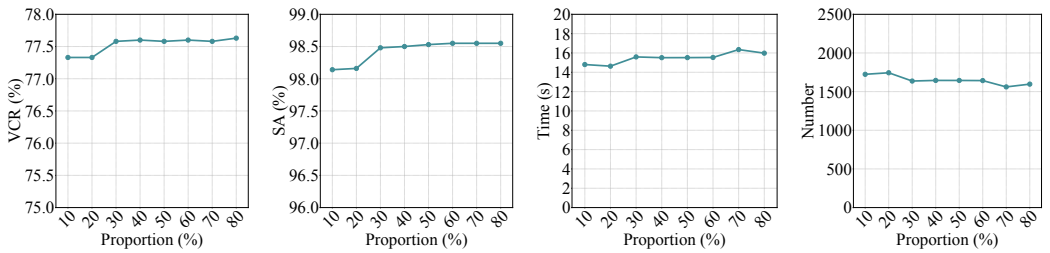
(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 11. Experimental results of varying correlation threshold λ.

VCR metric slowly increases at first, and then gradually decreases. According to Fig. 11(b), as the threshold, λ, increases, the SA metric initially shows an increase, and subsequently stabilizes.

Therefore, a larger correlation threshold, λ, can more effectively ensures that alerts triggered by different faults are not summarized into the same incident. However, an excessively large threshold may lead to alerts triggered by the same fault being summarized into different incidents. In our experiments, the optimal performance of ProAlert is achieved when λ is set to 0.88. Fig. 11(c) and Fig. 11(d) demonstrate that λ has little impact on the efficiency of ProAlert.

4.6.8 RQ8: Robustness to system updates. To evaluate the impact of system updates on ProAlert and assess its robustness, we maintain a fixed test set (the last 20% of alerts) while varying the proportion of alerts in the training set (from the first 10% to 80%). This experiment also reflect the ability of ProAlert to summarize alerts under different update intervals for its patterns. Since the business system corresponding to S1 experienced insignificant updates during the data collection period, we use S2 to assess the performance of ProAlert. Fig. 12 presents the experimental results.



(a) Valid compression ratio. (b) Summarization accuracy. (c) Time cost of alert summarization. (d) Average number of alerts processed per second.

Fig. 12. Experimental results of varying the proportion of training alerts.

We can find that VCR and SA exhibit improvement when the training set proportion reaches 40%. This indicates that during the time period corresponding to the first 30% to 40% of alerts, system updates notably affect fault propagation patterns. Therefore, relying solely on the first 30% of alerts is insufficient to capture corresponding patterns. However, the improvement of ProAlert at the 40% alert proportion is modest (only 0.25%). In addition, despite some fluctuations, ProAlert remains generally stable at other training set proportions. In terms of efficiency, we can observe that the proportion of training data has a limited impact on the efficiency of ProAlert.

The robustness of ProAlert stems from its use of a semantic-based clustering strategy to identify fault propagation patterns across different types of system components. Thus, unless a system update changes the type of system components or the fault types recorded in alert semantics, the mined patterns maintain a certain level of robustness.

Many system updates do not necessarily alter the types of system components or fault types. For example, a previously mined pattern indicating that disk-type faults propagate between disk components and server hardware components remains applicable even after a system update introduces a new disk device. While ProAlert demonstrates robustness to system updates, to address potential significant updates that could render existing patterns obsolete, we also recommend periodically updating the propagation patterns (e.g., weekly or daily).

5 Discussion

In this section, we present a real case to demonstrate the advantages of ProAlert over existing approaches and discuss the threats to the validity of our work.

5.1 Case Study

Table 4. An alert snippet from a real production environment

No.	Timestamp	Component	Content
a_{10}	2023/2/18 11:40:33	Host-5	The job with an error: rjob.sh-1, error type: file splitting.
a_{11}	2023/2/18 11:40:51	Storage-2	Used capacity (18TB, 90%) has reached the capacity warning threshold (warning threshold (18TB, 90%).
a_{12}	2023/2/18 11:41:00	Host-5	The job with an error: rjob.sh-2, error type: file splitting.
a_{13}	**2023/2/18 11:50:23**	**Host-5**	**The server has restarted, latest value 00:00:24.**
a_{14}	**2023/2/18 11:51:29**	**App-2**	**The Java process in app-2 has no response, please check.**
a_{15}	**2023/2/18 11:51:32**	**App-3**	**The Java process in app-3 has no response, please check.**
a_{16}	**2023/2/18 11:51:55**	**App-4**	**The Java process in app-4 has no response, please check.**
a_{17}	**2023/2/18 11:52:11**	**Server-4**	**Serial number Y006831, 2023/2/18 11:42:05 PwrOk Sig. Drop V_VDDQ_ABC power failure caused abnormal system shutdown.**
a_{18}	2023/2/18 11:52:31	Host-5	The job with an error: rjob.sh-1, error type: file splitting.
a_{19}	2023/2/18 11:52:42	Host-5	The job with an error: rjob.sh-2, error type: file splitting.
a_{20}	2023/2/18 11:52:49	Host-5	The job with an error: rjob.sh-3, error type: file splitting.

We present a real case from our experimental results to demonstrate the advantages of ProAlert over existing approaches. Table 4 shows an anonymized real alert snippet, while Figure 13 illustrates the system component relationships in the case. Specifically, server-4 is a physical machine, host-5 is a virtual Linux operating system deployed on server-4, and storage-2 is a storage disk array used by host-5. App-2, app-3, and app-4 are three microservices deployed on host-5.

The alerts in Table 4 were triggered by two distinct faults: a storage disk space shortage and a power failure. Figure 14 depicts the propagation paths of the faults. The insufficient disk space of storage-2 (a_{11}) caused multiple periodic script jobs on host-5 failed to execute due to file splitting errors (a_{10}, a_{12}, a_{18}, a_{19}, and a_{20}). On 2023-02-18 11:42:05, a sudden power failure caused server-4 to crash (a_{17}). Subsequently, front-line maintenance engineers received an urgent notification, promptly restarted the server (a_{13}). However, because the server had just restarted, the processes of app-2, app-3, and app-4 had not yet initialized, leading to the monitoring mechanism detecting no response from these processes (a_{14}, a_{15}, and a_{16}).

Both before and after the server crash, the storage disk space shortage fault persisted, resulting in interleaved alerts for the two faults. Due to the close topological relationship between system

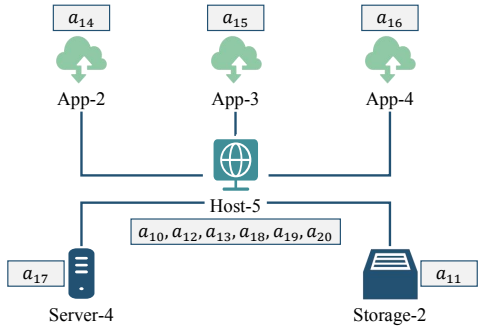


Fig. 13. The system topology of the case.

components in Table 4, existing approaches like LiDAR, OAS, and StormSum summarized these alerts into a single incident. In contrast, since ProAlert can deeply distinguish fault propagation patterns and verify fault propagation paths, ProAlert accurately identified that the alerts in Table 4 were triggered by different faults and summarized them into different incidents.

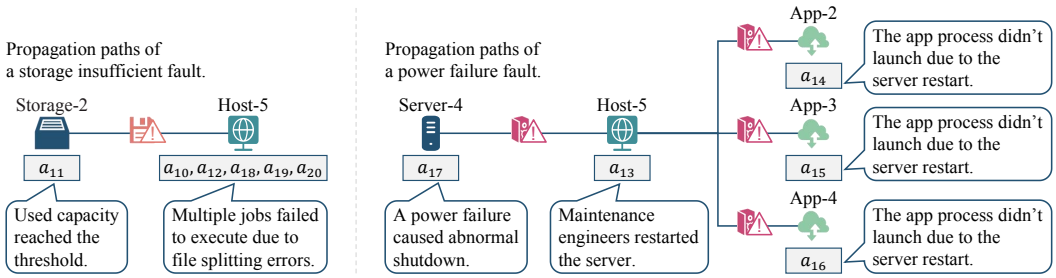


Fig. 14. The propagation paths of the faults in the case.

5.2 Error Analysis

There are two primary sources of errors in alert summarization: inaccuracies in mined patterns and mistakes in matched propagation paths. The error factors of mined patterns are as follows:

- Frequent Noise Alerts: In some systems, due to inadequate monitoring mechanisms, invalid noise alerts may be generated frequently. They interfere with pattern mining, leading to incorrect alert correlations. Thus, we recommend that operations engineers periodically inspect monitoring mechanisms and filtering out such invalid noise alerts.
- Insufficient Training Data: A limited training dataset may fail to capture the full spectrum of alert correlations. To mitigate this, we suggest utilizing the most current and comprehensive alert dataset available, coupled with periodic updates to the pattern mining process.
- Incomplete Alert Semantics: When alert contents lack precise descriptions, low semantic similarity between correlated alerts may hinder aggregation. Adopting standardized protocols requiring accurate fault symptom recording can address this issue.

The error factors of matched propagation paths are as follows:

- Inaccurate Embedding: If the embedding model fails to capture semantic nuances, it may result in incorrect matching of propagation paths. This can lead to incorrect matching of propagation paths. Thus, selecting or fine-tuning an appropriate model is crucial.

- Suboptimal Clustering: The effectiveness of clustering algorithms is affected by the selection of parameters. Therefore, careful calibration of clustering parameters is imperative for accurate incident detection.
- Missing or Incorrect Topology: While many organizations prioritize system topology maintenance, some may neglect its accuracy and completeness. Despite the ongoing effort required, topology is essential for system operations. Thus, we recommend continuous and systematic maintenance of online service system topology.

5.3 Threats to Validity

Threats to internal validity arise mainly from the implementation and parameter settings of experimental approaches, as well as the potential noise in manual labeling process. While we use the OAS source code directly, we implement LiDAR and StormSum ourselves due to the lack of public implementations. To mitigate this threat, we strictly follow the approaches outlined in their papers and select optimal parameters for summarizing the alerts of the experimental dataset.

Although ProAlert and StormSum do not require labeled data, both LiDAR and OAS necessitate labeled data for training. Therefore, we engage experts to label the correlations between alerts in the training data. However, given the high volume of alerts and interleaved faults, manually identifying all possible alert correlations is labor-intensive.

To reduce the workload of experts, we first summarize the training data using the unsupervised approach, ProAlert. Then, experts review the accuracy of each resulting incident and split incorrect incidents into multiple smaller, correct ones. Finally, overlapping incidents are automatically merged, and experts manually merge non-overlapping incidents that occur within a short time frame (within an hour) and belong to the same fault. This process ensures both the accuracy and completeness of the training data labeling.

Additionally, for the patterns mined by ProAlert and the incidents obtained from experimental approaches on the test data, we also engage experts to label their correctness. To ensure labeling accuracy, all manual labeling steps involve two independent experts, with conflicts resolved by a third expert via majority vote. Despite the potential for introducing noise during manual labeling, the experts are experienced and have rich domain knowledge. Therefore, we are confident that the amount of noises in labeling is small (if it exists).

The threats to external validity mainly lie in the selection of study subjects. In our experimental studies, we collected one month of alerts from a commercial company, A, which might limit the diversity of our dataset and the generalizability of our approach. Nevertheless, A is a very large commercial financial company serving over a billion users across hundreds of countries. Consequently, its service system is a typical, representative online service system generating sufficiently complex data. The performance of ProAlert on the real-world data of A indicates that ProAlert is generalizable enough and can benefit other companies.

6 Data Availability

The source code of ProAlert is available at https://github.com/Pro-Alert/ProAlert. In accordance with data confidentiality regulations, we are unable to make the dataset publicly available.

7 Related Work

Automated and intelligent approaches are crucial for enhancing system reliability and availability. Many studies focus on optimizing diagnosing faults in maintenance process from various aspects.

For alert summarizing, Lin et al. [17] leveraged unsupervised machine learning to effectively cluster the semi-structured alert text generated by IT infrastructure. Zhao et al. [32] conduct the first empirical study on the alert storm problem in online service systems, propose a novel method

for accurately detecting and summarizing alert storms and recommend representative alarms to maintenance engineers. Chen et al. [5] propose the LiDAR framework, to identify potential relationships between alerts, LiDAR integrates deep learning and natural language processing to analyze alert texts and system component dependencies. Chen et al. [3] employ supervised learning to extract semantic and behavioral information from alerts, automatically summarize alerts to improve alert processing efficiency. The DyAlert introduced by Chen et al. [7]is a approach for alert linking prediction based on dynamic graph neural networks, which enhances link accuracy by considering alert propagation information. However, it assumes that alerts are triggered by specific monitoring KPIs, a premise that may not always be applicable in real-world scenarios.

For incident triage, Chen et al. [1] conducted an empirical analysis of 20 large-scale online service systems at Microsoft. Their findings emphasize the importance of accurate incident triage, and point out both the potential and limitations of existing bug triage techniques in the context of incident triage.Building on this foundation, Chen et al. [2] propose DeepCT, an automated incident triage approach based on deep learning. This approach integrates a GRU model, attention mechanism, and a modified loss function to significantly enhance the accuracy and efficiency of incident triage in large online service systems.

For incident prediction, Chen et al. [6] propose AirAlert, a approach that leverages Bayesian networks and gradient boosting tree classifiers to predict and diagnose service outages in cloud service systems. On the other hand, the eWarn approach proposed by Zhao et al. [33] utilizes both historical and real-time alert data. It employs multi-instance learning and an interpretable machine learning model to offer an efficient and explainable incident prediction solution for online service systems. eWarn has been evaluated across 11 real-world online service systems and is applied in two major commercial banks, showcasing its effectiveness and practicality in actual use.

For incident analysis, there are some studies that utilize Large Language Models (LLMs) to automate the incident analysis of service outages. Jin et al. [16] introduced Oasis, a system that utilizes LLMs for automated evaluation of the impact range of service outages. In addition, Oasis generates easily comprehensible summaries, thereby enriching the comprehension of service outages. Chen et al. [4] introduce RCACopilot, a system that utilizes LLMs to automate the process of root cause analysis for service incidents. RCACopilot collects essential runtime diagnostic information and predicts the category of the root cause, leading to significant enhancements in both efficiency and accuracy in handling service incidents.

8 Conclusion

In this paper, we propose ProAlert, a novel approach that summarize alerts online via validating the propagation paths of faults. ProAlert learns the propagation patterns of faults between system components from history alerts in an unsupervised manner. These learned patterns are then used to validate the propagation paths of faults indicated by new alerts, thereby accurately summarizing alerts into incidents. Extensive experiments on real-world data demonstrate that ProAlert outperforms state-of-the-art approaches in alert summarization.

Despite the superior performance of ProAlert, it has some limitations. In the future, more effort can be devoted to the following directions:

- Improving the approach for mining propagation patterns of faults to enable ProAlert to respond in real-time to changes in fault propagation patterns and system topology.
- Enhancing ProAlert to not only identify the propagation paths of faults indicated by alerts but also determine the direction of fault propagation. Consequently, ProAlert can achieve both the online summarization of alerts and the automatic root cause localization of faults.

References

[1] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An Empirical Investigation of Incident Triage for Online Service Systems. In *2019 IEEE/ACM 41st International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. 111–120.

[2] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous Incident Triage for Large-Scale Online Service Systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 364–375.

[3] Jia Chen, Peng Wang, and Wei Wang. 2022. Online summarizing alerts through semantic and behavior information. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1646–1657.

[4] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, Jun Zeng, Supriyo Ghosh, Xuchao Zhang, Chaoyun Zhang, Qingwei Lin, Saravan Rajmohan, Dongmei Zhang, and Tianyin Xu. 2024. Automatic Root Cause Analysis via Large Language Models for Cloud Incidents. In *Proceedings of the Nineteenth European Conference on Computer Systems (EuroSys '24)*. Association for Computing Machinery, New York, NY, USA, 674â–688.

[5] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. 2020. Identifying linked incidents in large-scale online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 304–314.

[6] Yujun Chen, Xian Yang, Qingwei Lin, Hongyu Zhang, Feng Gao, Zhangwei Xu, Yingnong Dang, Dongmei Zhang, Hang Dong, Yong Xu, Hao Li, and Yu Kang. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *The World Wide Web Conference* (San Francisco, CA, USA) *(WWW '19)*. Association for Computing Machinery, New York, NY, USA, 2659–2665.

[7] Yiru Chen, Chenxi Zhang, Zhen Dong, Dingyu Yang, Xin Peng, Jiayu Ou, Hong Yang, Zheshun Wu, Xiaojun Qu, and Wei Li. 2023. Dynamic Graph Neural Networks-Based Alert Link Prediction for Online Service Systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 79–90.

[8] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xuemin Wen, Xiao Ling, Yongqiang Yang, and Michael R. Lyu. 2022. Graph-based incident aggregation for large-scale online service systems. In *Proceedings of the 36th IEEE/ACM International Conference on Automated Software Engineering* (Melbourne, Australia) *(ASE '21)*. IEEE Press, 430–442.

[9] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *North American Chapter of the Association for Computational Linguistics*.

[10] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1285–1298.

[11] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining* (Portland, Oregon) *(KDD'96)*. AAAI Press, 226–231.

[12] Alberto Fernandez and Sergio Gomez. 2008. Solving Non-Uniqueness in Agglomerative Hierarchical Clustering Using Multidendrograms. *Journal of Classification* 25 (02 2008), 43–65.

[13] Shangbin Han, Qianhong Wu, Han Zhang, Bo Qin, Jiankun Hu, Xingang Shi, Linfeng Liu, and Xia Yin. 2021. Log-Based Anomaly Detection With Robust Feature Extraction and Online Learning. *IEEE Transactions on Information Forensics and Security* 16 (2021), 2300–2311.

[14] Pinjia He, Jieming Zhu, Zibin Zheng, and Michael R. Lyu. 2017. Drain: An Online Log Parsing Approach with Fixed Depth Tree. In *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 33–40.

[15] Shilin He, Qingwei Lin, Jian-Guang Lou, Hongyu Zhang, Michael R. Lyu, and Dongmei Zhang. 2018. Identifying Impactful Service System Problems via Log Analysis. In *Proceedings of the 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 60–70.

[16] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, Shilin He, Federica Sarro, Yingnong Dang, Saravan Rajmohan, Qingwei Lin, and Dongmei Zhang. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE 2023)*. Association for Computing Machinery, New York, NY, USA, 1657â–1668.

[17] Derek Lin, Rashmi Raghu, Vivek Ramamurthy, Jin Yu, Regunathan Radhakrishnan, and Joseph Fernandez. 2014. Unveiling clusters of events for alert and incident management in large-scale enterprise it. Association for Computing

Machinery, New York, NY, USA.

[18] Qingwei Lin, Hongyu Zhang, Jian-Guang Lou, Yu Zhang, and Xuewei Chen. 2016. Log Clustering Based Problem Identification for Online Service Systems. In *Proceedings of the 38th International Conference on Software Engineering Companion*. ACM, New York, NY, USA, 102–111.

[19] Jinyang Liu, Zhihan Jiang, Jiazhen Gu, Junjie Huang, Zhuangbin Chen, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2023. Prism: Revealing Hidden Functional Clusters from Massive Instances in Cloud Systems. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. 268–280.

[20] Ping Liu, Haowen Xu, Qianyu Ouyang, Rui Jiao, Zhekang Chen, Shenglin Zhang, Jiahai Yang, Linlin Mo, Jice Zeng, Wenman Xue, and Dan Pei. 2020. Unsupervised Detection of Microservice Trace Anomalies through Service-Level Deep Bayesian Networks. In *IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 48–58.

[21] James MacQueen et al. 1967. Some methods for classification and analysis of multivariate observations. In *Proceedings of the fifth Berkeley symposium on mathematical statistics and probability*, Vol. 1. Oakland, CA, USA, 281–297.

[22] Weibin Meng, Ying Liu, Yichen Zhu, Shenglin Zhang, Dan Pei, Yuqing Liu, Yihao Chen, Ruizhi Zhang, Shimin Tao, Pei Sun, and Rong Zhou. 2019. LogAnomaly: Unsupervised Detection of Sequential and Quantitative Anomalies in Unstructured Logs. In *Proceedings of the 28th International Joint Conference on Artificial Intelligence*. IJCAI Organization, 4739–4745.

[23] Tomas Mikolov, Kai Chen, Gregory S. Corrado, and Jeffrey Dean. 2013. Efficient Estimation of Word Representations in Vector Space. In *International Conference on Learning Representations*.

[24] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2018. Advances in Pre-Training Distributed Word Representations. In *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. European Language Resources Association (ELRA), Miyazaki, Japan.

[25] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly Detection and Classification using Distributed Tracing and Deep Learning. In *19th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (CCGRID)*. IEEE, 241–250.

[26] Sasho Nedelkoski, Jorge Cardoso, and Odej Kao. 2019. Anomaly Detection from System Tracing Data Using Multimodal Deep Learning. In *IEEE 12th International Conference on Cloud Computing (CLOUD)*. IEEE, 179–186.

[27] Szymon Niewiadomski and Grzegorz Mzyk. 2023. ML Support forÂ Conformity Checks inÂ CMDB-Like Databases. In *Artificial Intelligence and Soft Computing*, Leszek Rutkowski, Rafał Scherer, Marcin Korytkowski, Witold Pedrycz, Ryszard Tadeusiewicz, and Jacek M. Zurada (Eds.). Springer Nature Switzerland, Cham, 366–376.

[28] Lior Rokach and Oded Maimon. 2005. *Clustering Methods*. Springer US, Boston, MA, 321–352.

[29] Shitao Xiao, Zheng Liu, Peitian Zhang, and Niklas Muennighoff. 2023. C-Pack: Packaged Resources To Advance General Chinese Embedding. arXiv:2309.07597 [cs.CL]

[30] Haowen Xu, Wenxiao Chen, Nengwen Zhao, Zeyan Li, Jiahao Bu, Zhihan Li, Ying Liu, Youjian Zhao, Dan Pei, Yang Feng, Jie Chen, Zhaogang Wang, and Honglin Qiao. 2018. Unsupervised Anomaly Detection via Variational Auto-Encoder for Seasonal KPIs in Web Applications. In *Proceedings of the 2019 World Wide Web Conference*. ACM, Republic and Canton of Geneva, CHE, 187–196.

[31] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, Junjie Chen, Xiaoting He, Randolph Yao, Jian-Guang Lou, Murali Chintalapati, Furao Shen, and Dongmei Zhang. 2019. Robust Log-Based Anomaly Detection on Unstable Log Data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 807–817.

[32] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and handling alert storm for online service systems. In *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering: Software Engineering in Practice* (Seoul, South Korea) *(ICSE-SEIP '20)*. Association for Computing Machinery, New York, NY, USA, 162–171.

[33] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-time incident prediction for online service systems. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering* (Virtual Event, USA) *(ESEC/FSE 2020)*. Association for Computing Machinery, New York, NY, USA, 315–326.

[34] Nengwen Zhao, Jing Zhu, Rong Liu, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Label-Less: A Semi-Automatic Labelling Tool for KPI Anomalies. In *IEEE Conference on Computer Communications*. IEEE, 1882–1890.

[35] Nengwen Zhao, Jing Zhu, Yao Wang, Minghua Ma, Wenchi Zhang, Dapeng Liu, Ming Zhang, and Dan Pei. 2019. Automatic and Generic Periodicity Adaptation for KPI Anomaly Detection. *IEEE Transactions on Network and Service Management* 16, 3 (2019), 1170–1183.

[36] Xiang Zhou, Xin Peng, Tao Xie, Jun Sun, Chao Ji, Dewei Liu, Qilin Xiang, and Chuan He. 2019. Latent Error Prediction and Fault Localization for Microservice Applications by Learning from System Trace Logs. In *Proceedings of the 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. ACM, New York, NY, USA, 683–694.

[37] Jieming Zhu, Shilin He, Jinyang Liu, Pinjia He, Qi Xie, Zibin Zheng, and Michael R. Lyu. 2019. Tools and benchmarks for automated log parsing. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 121–130.

Received 2024-09-13; accepted 2025-04-01