# FaultProfIT: Hierarchical Fault Profiling of Incident Tickets in Large-scale Cloud Systems

Junjie Huang
The Chinese University of Hong Kong
Hong Kong, China

Jinyang Liu
The Chinese University of Hong Kong
Hong Kong, China

Zhuangbin Chen
Sun Yat-sen University
China

Zhihan Jiang
The Chinese University of Hong Kong
Hong Kong, China

Yichen Li
The Chinese University of Hong Kong
Hong Kong, China

Jiazhen Gu*
The Chinese University of Hong Kong
Hong Kong, China

Cong Feng
Zengyin Yang
Computing and Networking
Innovation Lab, Huawei Cloud
Computing Technology Co., Ltd
China

Yongqiang Yang
Computing and Networking
Innovation Lab, Huawei Cloud
Computing Technology Co., Ltd
China

Michael R. Lyu
The Chinese University of Hong Kong
Hong Kong, China

## ABSTRACT

Postmortem analysis is essential in the management of incidents within cloud systems, which provides valuable insights to improve system's reliability and robustness. At CloudA[1], *fault pattern profiling* is performed during the postmortem phase, which involves the classification of incidents' faults into unique categories, referred to as *fault pattern*. By aggregating and analyzing these fault patterns, engineers can discern common faults, vulnerable components and emerging fault trends. However, this process is currently conducted by manual labeling, which has inherent drawbacks. On the one hand, the sheer volume of incidents means only the most severe ones are analyzed, causing a skewed overview of fault patterns. On the other hand, the complexity of the task demands extensive domain knowledge, which leads to errors and inconsistencies.

To address these limitations, we propose an automated approach, named FaultProfIT, for **Fault** pattern **Prof**iling of **I**ncident **T**ickets. It leverages hierarchy-guided contrastive learning to train a hierarchy-aware incident encoder and predicts fault patterns with enhanced incident representations. We evaluate FaultProfIT using the production incidents from CloudA. The results demonstrate that FaultProfIT outperforms state-of-the-art methods. Our ablation study and analysis also verify the effectiveness of hierarchy-guided contrastive learning. Additionally, we have deployed FaultProfIT at CloudA for six months. To date, FaultProfIT has analyzed 10,000+ incidents from 30+ cloud services, successfully revealing several fault trends that have informed system improvements.

## CCS CONCEPTS

• **Software and its engineering** → **Software evolution**; **Maintaining software**.

## KEYWORDS

Incident management, incident tickets, fault patterns

**ACM Reference Format:**
Junjie Huang, Jinyang Liu, Zhuangbin Chen, Zhihan Jiang, Yichen Li, Jiazhen Gu, Cong Feng, Zengyin Yang, Yongqiang Yang, and Michael R. Lyu. 2024. FaultProfIT: Hierarchical Fault Profiling of Incident Tickets in Large-scale Cloud Systems. In *46th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3639477.3639754

## 1 INTRODUCTION

Production incidents, which represent unplanned service interruptions or performance degradation, are inevitable in large-scale cloud services [14, 50]. They could decrease customer satisfaction and cause huge economic losses [1, 14, 34]. To effectively manage these incidents, cloud vendors (*e.g.*, Amazon Web Service [2], Microsoft Azure [3], and Google Cloud Platform [4]) have developed incident management systems [21] for prompt incident detection, diagnosis, and resolution. In such systems, the details of an incident are typically documented in an *incident ticket* (see an example in Figure 1), including its title, symptom, and resolution status, *etc.* The ticket is then tracked and updated throughout the incident lifecycle until the issue is resolved [14]. In general, the entire lifecycle of an incident can be divided into two main phases, *i.e.*, *real-time response* and *postmortem analysis* [21]. The former aims to quickly mitigate the incident's impact upon its occurrence. After incident resolution, the latter retrospectively examines the tickets to gain valuable insights that can enhance future incident management [41, 50].

*Corresponding author.
[1]Due to the company policy, we anonymize the name as CloudA.


*ICSE-SEIP '24, April 14–20, 2024, Lisbon, Portugal*

ACM ISBN 979-8-4007-0501-4/24/04…$15.00
https://doi.org/10.1145/3639477.3639754

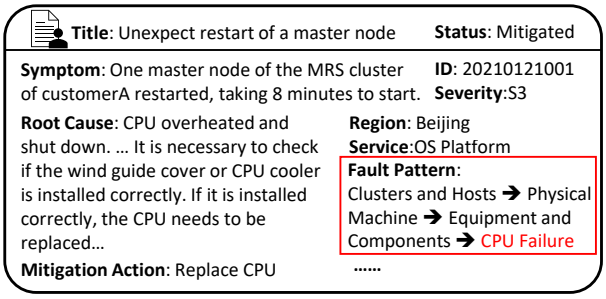


**Figure 1: An example of an incident ticket.**

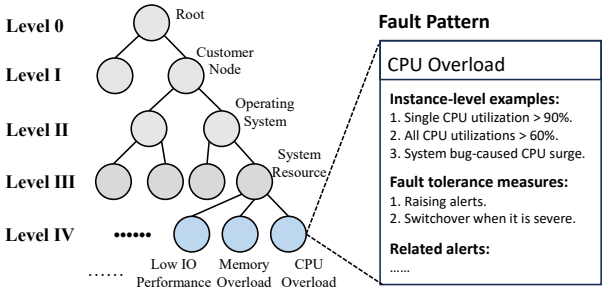


**Figure 2: Fault pattern example in the hierarchical taxonomy.**

Postmortem analysis plays an essential role in the continuous improvement of cloud systems' reliability and robustness [21, 24]. Specifically, it is conducted to understand the root cause of the incident, assess the impact, and evaluate the mitigation process, which can potentially be used to prevent similar incidents from happening again. Existing studies have demonstrated that production incidents could be recurring [36, 46] or share certain similarities [13]. Thus, the knowledge and insights derived from the postmortem analysis often exhibit recurring patterns. By aggregating and categorizing such recurring patterns, we can identify common faults, solutions, vulnerable components, and trends in the large volume of incidents, which can serve as a reference guide for understanding, diagnosing, and resolving future incidents more efficiently.

Based on this fact, the reliability engineers of CloudA (a top-tier cloud vendor offering global online services) perform the task of *fault pattern profiling* during postmortem analysis. This task involves classifying the faults that occurred during incidents into distinct categories, such as CPU overload, power outage, SSD failure, *etc.* We refer to each category as a *fault pattern*, which is a concise representation of the fault, including a fault name, a set of typical phenomena seen in historical examples, a list of possible mitigation measures, *etc.* A fault pattern of CPU failure is illustrated in Figure 1, which describes the breakdown of a physical machine in a cluster due to overheating. Replacing the CPU mitigates the issue. Clearly, the fault pattern offers readily information about the symptom and root cause of the incident together with actionable suggestions, which significantly accelerates the incident management pipeline. Due to the large scale and complexity of cloud systems, there exist a tremendous number of fault patterns. To better manage and exploit this knowledge, they are organized as a tree-like taxonomy based on their position across the entire cloud system stack (see Section 2.2). Figure 2 presents an example of such hierarchical taxonomy with five levels, which shows a customer node suffering a CPU overload issue. At CloudA, engineers value the role of fault patterns and have accumulated 334 of them.

In current practice, fault pattern profiling is carried out manually. This procedure involves carefully examining the tickets, identifying useful information, and aligning with the fault pattern taxonomy. While the manual approach is effective, it is time-consuming and prone to human error. First, the overwhelming volume of incidents implies that only a small fraction of incidents can be selected for in-depth postmortem analysis. This sampling may result in fault patterns that reflect only a partial distribution, thereby leading to a skewed overview and potentially sub-optimal improvement decisions. Second, the inherent complexity of the labeling task demands a deep understanding of the entire fault pattern hierarchy and the nature of incidents. Such requirements go beyond the ability of a single engineer. Thus, manual labeling will inevitably introduce errors and inconsistencies, leading to a distorted fault pattern distribution. Furthermore, the taxonomy of fault patterns is not static; it continuously evolves with the introduction of new patterns and adjustments in existing hierarchical relations. As a result, it is imperative to develop an automated approach for fault pattern profiling that can accommodate these complexities.

However, training a model capable of learning from existing fault patterns to automatically profile the unseen incident tickets is a non-trivial task, which presents the following two major challenges. First, fault patterns possess rich and complicated information, making their features hard to be exploited. As shown in Figure 2, each fault pattern not only has a structural position in the hierarchy but also includes explicit textual descriptions. The challenge lies in effectively harnessing such hybrid features, both hierarchical and textual, to accurately predict the fault patterns for unseen tickets. Second, the limited size of training samples poses another problem. The development of a robust fault pattern profiling model requires a substantial volume of labeled incident tickets. However, manually profiling fault patterns is both expensive and error-prone. Thus, only limited labeled examples are produced during the daily service maintenance at CloudA.

To address these challenges, we propose FaultProfIT, an automatic approach for **Fault** patterns **Prof**iling of **I**ncident **T**ickets based on hierarchical textual classification (HTC) [56]. Specifically, we employ hierarchy-guided contrastive learning [51] to train a hierarchical text classifier, aiming to precisely encode the sophisticated features of fault patterns while addressing the problem of data insufficiency. Contrastive learning has long been recognized as an effective way to learn meaningful textual representations [30, 44] with limited training samples by augmenting positive and negative samples and distinguishing among them [44]. By concentrating on similar input samples and pushing apart dissimilar ones, contrastive learning can enhance text representations and improve classification accuracy. In addition, to fully utilize the knowledge of fault patterns, we expand conventional contrastive learning to produce hierarchy-aware text representation. We apply an optimized Graphormer [55], a powerful graph representation model based on Transformer layers [48], to encode the hierarchical structures and node descriptions together. These representations capture both the semantics and hierarchy of fault patterns, and thus can support more accurate profiling.

We have evaluated and deployed FaultProfIT at CloudA. Our evaluation demonstrates that FaultProfIT achieves a high degree of accuracy (78.3% F1-score) in automatic profiling of fault patterns, outperforming a wide range of text classification models. We also conduct a comprehensive ablation study and analysis to demonstrate the effectiveness of our hierarchy-guided contrastive learning approach in learning hierarchy-aware incident representation. Furthermore, we profile fault patterns across incidents of various categories and find that some incidents, such as those associated with lighter severity and those from infrastructure and computing services, exhibit higher accuracy. Lastly, we have deployed FaultProfIT to a cloud reliability analysis system at CloudA, an incident management and analytics platform used by over 30 service teams and thousands of engineers. FaultProfIT has been running at CloudA for six months, and successfully identified a number of emerging fault trends, which in turn guide engineers to fix the vulnerabilities, thus improving system reliability.

To sum up, this paper makes the following contributions:

- To the best of our knowledge, we are the first to automatically profile fault patterns of incident tickets for postmortem analysis.
- We propose FaultProfIT, which leverages hierarchy-guided contrastive learning to learn hierarchy-aware incident representation to classify pattern pattern labels.
- We conduct an extensive evaluation on the production incidents at CloudA. The results show that FaultProfIT outperforms state-of-the-art methods.
- We have deployed FaultProfIT at CloudA for six months, where it has analyzed over 10,000 incidents from 30+ cloud services and revealed fault trends for system improvements.

## 2 BACKGROUND AND MOTIVATION

### 2.1 Incident and Incident Management

In cloud systems, an incident is defined as an unplanned interruption or performance degradation of a service or product that impacts service availability and customer satisfaction [14, 21]. For example, a slow connection, an unavailable service, and a customer-reported error could constitute an incident.

*2.1.1 Incident Lifecycle.* In order to accelerate incident mitigation and prevent the incident from happening again, cloud vendors such as CloudA build *incident management* systems to assist engineers during the whole incident life cycle [14, 46, 59]. Figure 3 shows a typical example of the incident lifecycle, which can be broadly divided into two phrases, *i.e.*, *real-time response* and *postmortem analysis*.

**Real-time Response.** When an incident occurs, On-Call Engineers (OCEs) must take immediate action to resolve the incident to minimize its impact. This process begins with *incident reporting*, where an incident is initially raised [22, 34]. In cloud systems, the incidents can be reported by customers when they encounter problems during service usage or detected by tailored system monitors when performance metrics fall below pre-defined acceptance levels. A severity level is also assessed to measure the impacts of each incident and determine whether additional investigation is required [5]. Then, in the *incident triage* stage, the incident will be routed to an appropriate service team for resolution [9]. Based on service ownership and heuristic algorithms, the responsible team is automatically determined. However, due to the high complexity and dependencies, the incident could be incorrectly triaged. In this case, it will be re-routed to a more appropriate team for investigation, and this process can repeat several times [20]. The final stage is *incident mitigation*, in which the service team investigates the incident and takes mitigation actions to bring the problematic service back to normal. In practice, some temporary workarounds (*e.g.*, server rebooting and configuration change) will be applied first to quickly mitigate the impact. But occasionally, the team can encounter intractable problems. In such cases, they can escalate and engage additional teams for investigation, which often necessitates more complex mitigation measures to cloud systems, such as bug fixing and version rollback. Upon the resolution of the incident, it will be closed in the incident management system. During the real-time response period, OCEs create incident reports as a record of diagnosis. The report is written by following some rigorous rules to ensure clarity, thereby facilitating subsequent diagnostic procedures and postmortem analyses. At CloudA, an incident report contains plentiful information, including an incident timeline, temporary root causes, escalating records, mitigation actions, *etc.*

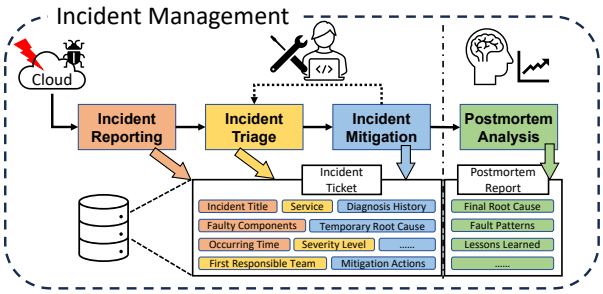


**Figure 3: The incident management process.**

**Postmortem Analysis.** At CloudA, when an incident is resolved, a *postmortem analysis* will often be conducted to evaluate the circumstances retrospectively. The insights gleaned from this analysis are invaluable, serving as important guidelines for the continuous improvement and enhancement of the cloud system's reliability. During analysis, engineers need to write a postmortem report to reflect the whole incident picture and summarize useful knowledge for future retrospection. At CloudA, a postmortem report derives from the original incident ticket and the reliability team will involve more contents to it such as (1) the final root causes of incidents, (2) fault categories to tag the incident, and (3) suggestions to system improvement. Such contents are maintained in natural language and categorical data, which are easily accessible and shareable. Due to the large volume of incidents and limited human resources, only a small sample of incidents will be selected and analyzed during postmortem. The selection criteria of incidents are ad-hoc, mainly based on engineer feedback, incident severity, and observed trends. Our work applies directly to enhancing postmortem analysis and contributing to continuous improvement in cloud reliability, which deals with automatic fault profiling for incident tickets.

*2.1.2 Incident Management System.* At CloudA, thousands of incidents are reported to OCEs every day from various sources such as external customers, internal engineers, and automated monitoring

**Table 1: Categories of fault patterns at level I.**

| Level I | Description | Fault Pattern Example |
|---|---|---|
| Infrastructure and Sites | Incidents that occur within the physical and network infrastructure of a site. This includes problems related to external and dedicated networks, data center environment and facilities, and data center network equipment. | Infrastructure and Sites → Data Center Environment → Data Center Facilities → Power Supply Insufficient |
| Clusters and Hosts | Incidents that occur within the system clusters and specific hosts, including the physical machines, virtual machines, containers, and storage. | Clusters and Hosts → Physical Machine → Device and Components → SSD Failure |
| Customer Node | Incidents that specifically occur within the business nodes of customers, impacting the operating system and its processes or threads. | Customer Node → Operating System → System Resources → CPU Overload |
| Load and Capacity | Incidents disrupt the balance and efficiency of system load and capacity management, leading to performance degradation, traffic surges, security threats, and resource allocation issues. | Load and Capacity → Overload Control → Frontend Load → Security Attacks |
| Business and Data | Incidents that occur during the operation and management of business processes and data, including problems with tenant resources, business configurations, licensing, security credentials, and system configurations. | Business and Data → Tenant Resources → Tenant Account and Data → Tenant Data Deletion |
| Dependencies | Incidents that occur within the internal and external dependencies of the system, including problems with web servers, databases, microservices, and cloud service dependencies. | Dependencies → Internal Dependency → Database → Unavailable Database Service |
| Disaster Recovery | System's inability to recover and resume normal operations after a significant disruption over regions or availability zones. | Disaster Recovery → AZ Disaster Recovery → AZ Site → Active AZ Site Failure |

systems. To manage the incidents at scale, CloudA developed a web application for company-wide incident reporting, investigation, and analysis. During an incident's lifecycle, an incident ticket is well-documented to record the incident-related information by various participants, such as OCEs to verify and report the incidents, SREs to mitigate the incidents, and cloud reliability teams to conduct postmortem analysis and derive valuable insights. Figure 1 shows an example of incident tickets. These tickets contain fruitful information such as incident timelines, severity, summaries, root causes, and mitigation suggestions written by SREs.

## 2.2 Fault Pattern Profiling

*Fault pattern profiling* is a crucial task employed by the reliability team at CloudA to derive insights from postmortem analysis. This task involves classifying the faults that occurred during incidents into distinct categories, which is referred to as *fault pattern*.

**Fault Pattern.** At CloudA, a fault pattern characterizes abnormal behaviors exhibited in specific objects. Each fault pattern comprises a fault name, a set of potential phenomena, measures for fault tolerance, *etc.* An example of a fault pattern, as shown in the right segment of Figure 2, is *CPU overload*. This fault pattern is described by phenomena such as a single CPU utilization exceeding 90%, and fault tolerance strategies such as switchover. The concise representation of faults as fault patterns enables engineers to readily understand the nature of the faults, thereby facilitating efficient fault diagnosis and mitigation.

**Fault Pattern Taxonomy.** The reliability team has developed a comprehensive *fault pattern taxonomy* to manage a multitude of fault patterns across diverse objects. This taxonomy is structured in a tree-like hierarchy with five levels, comprising 7 hyper classes at level I and 334 fault patterns at leaf nodes. For example, as shown in the left segment of Figure 2, a *Customer Node* consists of the Operating System level and the Process level, which can be further subdivided into system resource, environment, and so on.

The principle of constructing the taxonomy is to divide and group fault patterns based on the specific components where the faults occur. In practice, it was initiated by analyzing historical incident records to identify common patterns. Similar fault phenomena in the same object were summarized into a single fault pattern. Subsequently, similar fault patterns were grouped into a hyper-class based on the specific objects in which they all occur. The hyper-class can broadly contain region-level or az level components, but it can also be narrowed down to a VM or a system environment. For example, CPU overload, memory overload, and low IO performance all reflect different aspects of system resources. With the dedicated partitioning, the reliability produced the first version of the taxonomy, which has been maintained for over eight years and has undergone multiple rounds of refinement. It is now considered comprehensive and ready-to-use, and is continuously updated to accommodate new incidents and fault patterns. Table 1 shows the seven top-level categories in the taxonomy, each consisting of a brief description and multiple finer-grained subcategories covering faults in various system components. The fault pattern taxonomy is a valuable knowledge base of great utility, which has also been successfully applied in other reliability scenarios at CloudA, such as guiding fault injection and informing disaster recovery design.

**Automatic Fault Pattern Profiling.** A crucial aspect of postmortem analysis at CloudA involves the examination of incident fault patterns. The objective of this process is to categorize incidents for follow-up analysis. Figure 1 shows an incident and its associated fault pattern. By evaluating the distribution of fault patterns over a specific timeframe, the reliability team can discern system trends and recurring faults. This data-driven understanding can guide strategic business decisions and assist in setting overarching group targets. Additionally, engineers can leverage fault pattern categories to retrieve relevant incidents of a similar nature, thereby serving as a valuable reference during fault diagnosis and a knowledge repository for experience sharing.

The current practice of fault pattern profiling is conducted through manual labeling during the postmortem phase. Reliability engineers first examine the diagnostic details within the tickets and then identify anomalous behaviors, which are typically indicated in natural language in the tickets. Subsequently, they align the information in fault patterns with the incident to determine fault patterns. Practically, a single incident can exhibit multiple fault patterns, indicating the simultaneous occurrence of multiple faults.

Despite the effectiveness of manual profiling, it is labor-intensive and susceptible to errors, potentially yielding biased insights regarding system maintenance. Firstly, the overwhelming volume of incidents means that only a small fraction of these incidents undergo postmortem analysis due to the limited human resources and high cost of analysis [18]. Incidents of high severity are prioritized, resulting in less critical ones being neglected, thereby making partial coverage. For example, Figure 4 shows the fault pattern distribution with respect to incidents of different severity that has been analyzed during postmortem. The incidents with severity S3 significantly outnumber less severe ones. However, in reality, incidents in S4 and S5 should be more prevalent. Secondly, the assignment of fault pattern types requires extensive domain knowledge of incidents and fault patterns. However, the varying expertise levels of engineers can introduce errors and inconsistencies in manual profiling. For example, 29% root cause tags assigned by OCEs at Microsoft are incorrect [18].

To address these issues, our work introduces techniques to automate the profiling of fault patterns in incident tickets. Figure 5 shows an overview of the task. Our approach not only improves the efficiency of postmortem analysis but also provides more accurate fault patterns for downstream applications.

## 3 METHODOLOGY

As discussed before, manually profiling fault patterns for incident tickets is labor-intensive and error-prone, leading to a biased understanding of faults in cloud systems. To address the issue, we propose FaultProfIT, an automated tool for fault pattern profiling at CloudA. FaultProfIT utilizes language models to read the diagnostic descriptions in incident tickets and predicts the fault pattern labels from the taxonomic hierarchy, which can improve the efficiency of postmortem analysis and provide actionable insights for business decision making. In this section, we introduce our method in detail.

### 3.1 Overview

FaultProfIT utilizes hierarchical text classification techniques [51, 56] to predict fault patterns for incident tickets. The main concept of FaultProfIT is to employ pretrained language models (PLM) [17] to comprehend the semantics of incident tickets and incorporate a taxonomic hierarchy into the PLM to produce hierarchy-aware representations for classification. Figure 6 shows an overview of FaultProfIT. Given an incident ticket, we first extract relevant data from the ticket to establish the incident context (§ 3.2). Subsequently, we apply an incident encoder based on the MacBERT PLM [16] to encode incident context into vector features for classification (§ 3.3). The encoder is trained to incorporate hierarchical fault pattern information by the hierarchy-guided contrastive learning [51]. In contrastive learning, building challenging positive samples is crucial [44]. Therefore, guided by the taxonomic hierarchy (§ 3.4), we construct high-quality positive samples that are both label-involved and hierarchy-aware for the incident context (§ 3.5). By pulling closer to the original incident contexts with augmented samples, the incident encoder can learn to generate hierarchy-aware textual representations (§ 3.6). Finally, after training, FaultProfIT can discard the redundant hierarchy and utilize the hierarchy-enhanced incident encoder to classify fault patterns.

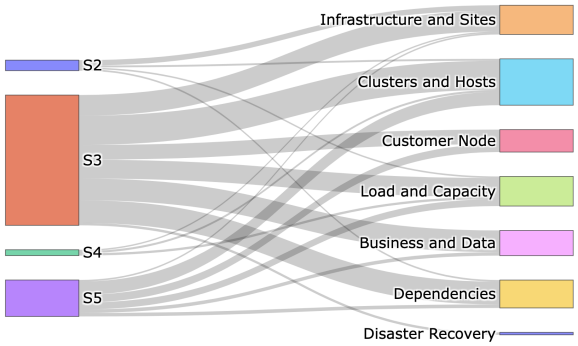


**Figure 4: Fault pattern distribution of incidents with different severities that have undergone postmortem.**

### 3.2 Incident Data Fetching and Preprocessing

During the incident lifecycle, different groups of engineers collaboratively contribute to different fields of the incident tickets at different stages. In order to help the postmortem process and prevent any data leakage, we assume only the fields of tickets before postmortem analysis can be available for fault pattern profiling (Figure 3 shows the typical fields before postmortem). In our work, we select four types of information from tickets to form the *incident context*, which is used for the following profiling, including incident title, symptoms, temporary root causes, and mitigation actions.

In most cases, OCEs do not follow specific formats to fill in the tickets. For example, the symptoms of tickets could be in various forms, such as textual descriptions, images, tables, runtime logs, and shell scripts. This is because the incidents are very different from each other, and the utmost priority of the OCEs is to mitigate the incident as soon as possible rather than carefully document the tickets. However, these multimodal contents may not be recognized by language models and can add additional noise to the vocabulary. Therefore, we conduct a series of *data cleaning* before feeding the tickets into language models. To deal with that, we first remove the multimodal information, including images and tables from the symptoms. Then, we conduct text preprocessing by removing URLs, HTML tags, and codes using regular expressions and parsers. In this process, we also clean up the text by removing extra spaces, new line marks, and extra braces. Finally, we concatenate the selected textual information by adding a brief description for each field to construct the incident context to the language model, which is shown below. The designed format can make the ticket contents more fluent and interpretable to language models, which is beneficial to improve the accuracy [39].

**Incident context:** Incident ticket title: [Title]. Symptoms of incidents: [Symptoms]. Identified root cause: [Temporary Root Cause]. Mitigation actions: [Mitigation Actions]

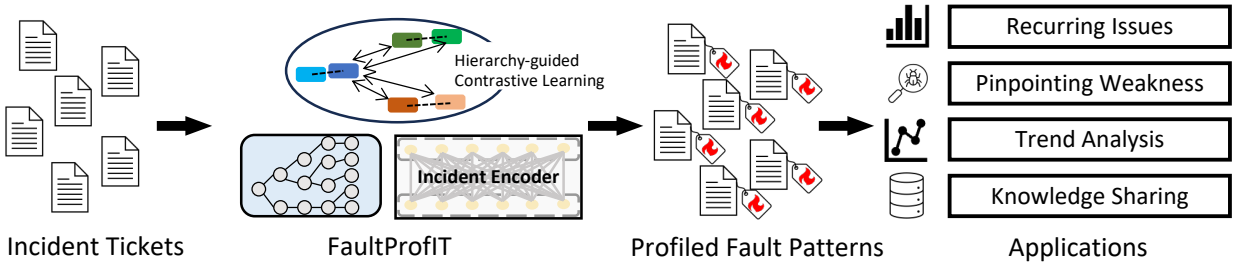


**Figure 5: The overview of automatic fault pattern profiling.**

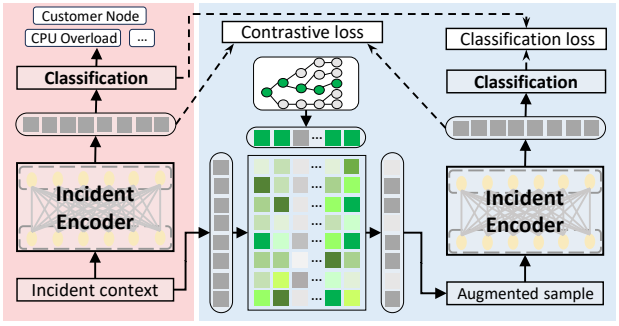


**Figure 6: The overview of FaultProfIT. Parts in blue and red denote the training and prediction workflow, respectively.**

## 3.3 Incident Encoder

The incident encoder aims to transform the raw text of incidents into representation vectors, which serve as features for classification. Our approach requires a strong encoder to represent the incident context due to the diverse contents in incident tickets. Therefore, we adopt pretrained language models (PLMs) [17] to encode incident context, which have demonstrated remarkable ability to understand the semantic meaning of incidents and have proven effective in recent incident understanding tasks [6, 28]. In this work, we leverage MacBERT [16] as our text encoder, which is a Transformer encoder model with the same architecture as BERT [17]. The reason we choose MacBERT over BERT is that MacBERT is an optimized version of BERT trained on the multi-lingual corpus and is capable of processing both Chinese and English. Formally, an input token sequence of incident context is represented as $x = \{\texttt{[CLS]}, x_1, x_2, \ldots, x_{n-2}, \texttt{[SEP]}\}$, where `[CLS]` and `[SEP]` are two special tokens to indicate the beginning and the end of the sequence. The input sequence is then mapped into input representations and fed into MacBERT to obtain the hidden representations for each token: $\mathbf{X} = \texttt{MacBERT}(x), \mathbf{X} \in \mathbb{R}^{n \times d_t}$, where $n$ is the number of tokens in $x$ and $d_t$ is the embedding dimension. We use the hidden representation of the first token (`[CLS]`) to represent the whole sequence $\mathbf{x} = \mathbf{X}_{\texttt{[CLS]}}$.

## 3.4 Hierarchy Encoder for Fault Patterns

The hierarchy encoder aims to transform the fault pattern hierarchy into a series of feature vectors, where each vector represents a node in the hierarchy. In our work, we formulate the fault pattern hierarchy as a Directed Acyclic Graph (DAG) $G = (F, E)$, where each node $f_i$ from the node set $F$ contains a node label and description, and the edge set $E$ represents a set of parent-child relations among pairs of nodes, *i.e.*, the overall hierarchy. Then we use an optimized Graphormer [55] to encode the graph, which is the state-of-the-art graph representation architecture based on Transformer layers [48] with spatial encoding and edge encoding.

Our hierarchy encoder first maps the nodes from the graph into a set of input feature vectors. The default Graphormer uses a randomly initialized label embedding of $f_i$ as the input vector $\mathbf{f}_i$. However, in our task, each node contains a clear natural language description of the fault label, which we believe provides fruitful information and can benefit the node representation. Therefore, we adopt the optimized input feature vector of a node to enrich the semantics, which is computed as the sum of label embedding and its description embedding:

$$\mathbf{f}_i = \mathit{LabelEmbedding}(f_i) + \mathit{DescriptionEmbedding}(f_i) \quad (1)$$

The label embedding is randomly initialized and learnable during training with a dimension of $d_t$. The description embedding is computed by a MacBERT encoder using the average of hidden token representations of the description, which also has a size of $d_t$. Specially, we obtain the textual description depending on the node type. For fault patterns in the leaf nodes, we simply concatenate the fault name, instance-level examples, and fault tolerance measures with a brief description for each field. For the fault node in the first four levels, we concatenate its name and description to form the textual input. Finally, we obtain all the $k$ input node feature vectors, which can be stacked as a matrix $\mathbf{F} \in \mathbb{R}^{k \times d_t}$.

After obtaining input node features, the hierarchy encoder injects the parent-child relations to obtain hierarchy-aware node representations with Graphormer architecture:

$$\mathbf{H} = \texttt{Graphormer}(\mathbf{F}) \quad (2)$$

Concretely, Graphormer encodes the structural information by spatial encoding and edge encoding. The edge encoding computes the aggregated weight of edges within the path of two nodes, and the spatial encoding measures the distance between two nodes, where both weight matrices are added into the original Query-Key product matrix in the self-attention layer. Here we omit the Graphormer architecture and inherent computation, and please refer to the original paper for the details.

## 3.5 Positive Sample Construction

A critical pre-step of contrastive learning is to build challenging positive samples that can be used to contrast [7, 25, 54]. In our

task, we aim to construct high-quality positive samples for each label considering the taxonomic hierarchy, which can guide models to acquire hierarchy-aware label representations [51] (Section 4.4 shows the obtained hierarchy-aware representations). The idea of our positive sample construction approach originates from an observation that when text is classified into a certain category, most words are unimportant [7]. For example, when an incident description is classified as "CPU overload", some keywords such as "CPU utilization" or "exceed 90%" provide strong signals while words like users or clusters have fewer impacts. Therefore, the perturbed text that removes some unimportant tokens while keeping the keywords should maintain the original label and can be regarded as a positive sample. Based on this observation, we construct positive samples and utilize fault pattern hierarchy to guide the keyword selection.

We locate the important keywords under a given label by computing the attention weights to the hierarchy-aware label representation and then gather the tokens with larger weights to build the positive samples. Concretely, given the hidden token representations $\mathbf{X} \in \mathbb{R}^{n,d_t}$ of an incident ticket, we first compute the scale-dot attention [48] to the node representation $\mathbf{H} \in \mathbb{R}^{k,d_t}$ to obtain the attention weight matrix $A \in \mathbb{R}^{n,k}$, where each element $A_{ij}$ determines the importance of the $i$-th token on a node label $f_j$. After that, we sample important tokens from the importance matrix for a given label $f_j$ to form a positive sample $\hat{x}$. To make the sampling differentiable, we replace the softmax function with gumbel-softmax [27] to calculate the probability that $x_i$ is the keyword of class $f_j$ by:

$$P_{ij} = gumbel\_softmax(A_{i1}, A_{i2}, \ldots, A_{ik})_j, \tag{3}$$

which satisfies $\sum_j P_{ij} = 1$. Since each incident can have multiple fault pattern labels, we obtain the final importance score of each token $x_i$ by simply adding up the probability of the token regarding its ground-truth label set $f$ as $P_i = \sum_{j \in f} p_{ij}$. The importance scores are used for the final positive sample construction. As one label can have multiple important tokens, we do not transform the probability to one-hot vectors after softmax for discretization. Instead, we keep the tokens for positive samples if their probabilities of being sampled exceed a threshold $\lambda$. Therefore, the positive sample $\hat{x}$ can be constructed as:

$$\hat{x} = \{x_i \texttt{ if } P_i > \lambda\} \tag{4}$$

The positive sample $\hat{x}$ is encoded by the same text encoder to obtain the hidden representations: $\hat{\mathbf{X}} = \texttt{MacBERT}(\hat{x})$. The hidden representation of first token ([CLS]) is used for sequence representation: $\hat{\mathbf{x}} = \hat{\mathbf{X}}_{\texttt{[CLS]}}$.

## 3.6 Contrastive Learning

After obtaining positive samples, we adopt contrastive learning [11] to train hierarchy-aware incident and fault pattern representations for better fault pattern profiling. Contrastive learning aims to learn representations by enforcing positive samples to be closer while keeping negative samples further apart. This is achieved by leveraging a contrastive loss function to maximize the similarities of positive samples within the batch and has been proven effective in learning strong representations in many classification tasks [43, 45, 52]. Specially, for each incident sample, we have one positive sample constructed by the method in Section 3.5, and $2(N-1)$ negative samples which are all the remaining samples except for $x$ and $\hat{x}$ in the training batch with a batch size of $N$. Finally, we compute the NT-Xent [11] contrastive loss function of all examples in the batch:

$$Loss^{contra} = -\sum_{i=1}^{2N} \log \frac{e^{\texttt{cosine}(\mathbf{x}^{(i)}, \hat{\mathbf{x}}^{(i)})/\tau}}{\sum_{j=1, j\neq i}^{2N} e^{\texttt{cosine}(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})/\tau}}, \tag{5}$$

where $\texttt{cosine}(\cdot,\cdot)$ is the cosine similarity between two vectors and $\tau$ is a temperature hyperparameter.

## 3.7 Classification and Training Objective

*Classification.* The classification module aims to map each incident feature vector into a set of labels. Following previous works [51, 60], we feed the incident representation $\mathbf{x}$ into a linear classifier with a sigmoid activation function for multi-label classification. The probability on fault labels $f_j$ is computed as:

$$p_j = \texttt{sigmoid}(W_c \cdot \mathbf{x} + b_c)_j, \tag{6}$$

where $W_c \in \mathbb{R}^{k \times d_h}$ and $b_c \in \mathbb{R}^{d_h}$ are the weights and the bias term. The labels with the probabilities exceeding a certain threshold will be collected as the prediction, which is set to 0.5 in our work. Notice that we train the incident encoder with hierarchy-guided contrastive learning, which is supposed to be injected with the knowledge of fault pattern hierarchy. Thus, we do not need to incorporate the node representation encoded by hierarchy fault pattern encoder during classification, which is more computationally efficient and effective.

*Training Objective.* During training, we jointly optimize all parameters of our model, including the incident encoder, hierarchy encoder, positive sample construction module, and the classification module. These components work together to categorize fault patterns and learn representations in a contrastive manner. For fault pattern profiling, we employ *weighted binary cross-entropy loss*, a commonly used loss function for multi-label classification. In this context, a weight parameter $\gamma$ is introduced:

$$Loss^{cls} = -\sum_{i=1}^{N}\sum_{j=1}^{k} \gamma f_j^{(i)} \log(p_j^{(i)}) + (1 - f_j^{(i)}) \log(1 - p_j^{(i)}). \tag{7}$$

The role of $\gamma$ is to mitigate class imbalance and enhance model's sensitivity towards infrequent classes. The objective is to minimize this loss, *i.e.*,, to make the predicted probability as close as possible to the true label.

In Section 3.5, we construct positive samples by keeping a few important tokens, which are supposed to maintain the original labels. To train a more robust classifier, we involve the constructed positive samples and compute the classification loss $\hat{Loss}^{cls}$. Similar to Equation 7, we adopt binary classification loss by substituting $p_j^{(i)}$ to $\hat{p}_j^{(i)}$, where the probability $\hat{p}_j^{(i)}$ can be obtained with the same classification module.

The final loss function is the combination of classification loss of original samples, classification loss of constructed positive samples, and contrastive learning loss function:

$$Loss = Loss^{cls} + \hat{Loss}^{cls} + \alpha Loss^{contra}, \tag{8}$$

where $\alpha$ is a hyperparameter to control the weight of $Loss^{contra}$.

# 4 EVALUATION

We evaluate our method by answering the following research questions (RQs):

- RQ1: How effective is FaultProfIT in fault pattern profiling?
- RQ2: How does hierarchy-guided contrastive learning affect FaultProfIT?
- RQ3: How does FaultProfIT perform on diverse types of incidents?

## 4.1 Experiment Designs

*Dataset.* We collect incident tickets from the incident management system at CloudA, which are created from January 1, 2017, to December 31, 2022. The system is used by a range of service teams such as computing, networking, and storage. As FaultProfIT is proposed to assist in postmortem analysis, we conduct filtering to obtain incidents that are ready for postmortem. In particular, we only include tickets from the "Mitigated" ones and remove those with all empty contents in the fields of symptom, temporary root cause, and mitigation actions. Finally, we collect 22,560 incidents and 1,463 with annotated fault pattern labels. The number of annotated samples is low since the profiling was done in the postmortem analysis stage, and the postmortem is not done for every incident.

We use the 1,463 incident tickets that are labeled with fault patterns to form our dataset for evaluation. The dataset is randomly split into training, valuation and test with a portion of 80%:10%:10%. We tune our model on the validation set to search for the best hyperparameters and report the results on the test set.

*Implementation Details.* We conduct our experiments on a Linux GPU server with Intel Xeon 2.3GHz CPU and NVIDIA Tesla T4 16G GPU. We implement FaultProfIT with Python 3.7.10, PyTorch 1.10.0 [42] and transformers 4.2.1 [53]. For Graphormer, we set the attention head to 8 and feature size $d_t$ to 768, which is the same dimension as the representations produced by the MacBERT encoder $d_h$. The maximum input token length of MacBERT is 512. During training, we use Adam [31] optimizer with a learning rate of 1e-5 and linear scheduling with 5% warm-up. We set the training batch size as 8 and train the model for 100 epochs. As for hyperparameters, the contrastive loss weight $\alpha$, binary classification weigh $\gamma$ and threshold $\lambda$ are selected by grid search on development set where $\alpha$ is set to 0.1, $\gamma$ is set to 5 and $\lambda$ is set to 0.01. The temperature $\tau$ of the contrastive module is set to 1.

## 4.2 RQ1: How effective is FaultProfIT in fault pattern profiling?

**Setup.** In this RQ, we evaluate the effectiveness of FaultProfIT on the fault pattern profiling task. We compare the whole FaultProfIT model against three baseline methods to classify fault patterns. We use the average precision, average recall and F1 score over all examples as the comparing metrics. Specifically, we compare FaultProfIT with the three following baseline methods.

- **Dense Passage Retriever** [29] (DPR) is the state-of-the-art text matching model, which relates the incident context to fault descriptions in a joint vector space to find the relevant fault patterns. We simply take the top-5 labels as predictions since taxonomy has five levels. The DPR is unaware of the hierarchy.
- **MacBERT** [16] is a multi-label text classifier that feeds the concatenation of incident context and fault descriptions into a MacBERT classifier to obtain relevance scores. It treats taxonomy as flattened labels without considering the structure.
- **ChatGLM** [19] is a bilingual (English and Chinese) large language model with 13B parameters. We tune ChatGLM to classify fault patterns among flatted labels without structure with a parameter-efficient tuning method based on P-Tuning [40] at INT4 quantization level.
- **HiAGM** [60] is a state-of-the-art hierarchical text classification model which matches incident context embeddings to fault pattern embeddings encoded by Graph Convolution Networks. HiAGM has leveraged the hierarchical relationships.

**Table 2: Experiment results of different models.**

| Method | Precision | Recall | F1-score |
|---|---|---|---|
| Dense Retriever | 48.5 | 61.1 | 54.1 |
| MacBERT | 58.5 | 61.9 | 60.1 |
| ChatGLM | 60.0 | 65.2 | 62.5 |
| HiAGM | 72.1 | 78.2 | 75.1 |
| **FaultProfIT** | **76.6** | **80.1** | **78.3** |

**Results.** Table 2 shows the performance of different models on fault pattern profiling. We observe that FaultProfIT performs substantially better than the other three baseline methods among all metrics, indicating the superiority of FaultProfIT in identifying correct fault patterns for incident tickets. In addition, models leveraging the taxonomic structures (*i.e.*, FaultProfIT and HiAGM) outperform models without considering the structures (*i.e.*, Dense Retriever, MacBERT, and ChatGLM) by a large margin. This result demonstrates the importance of injecting hierarchical information to guide models to capture label relationships for better profiling.

## 4.3 RQ2: How does hierarchy-guided contrastive learning affect FaultProfIT?

**Setup.** FaultProfIT leverages contrastive learning guided by taxonomic hierarchy to train hierarchical aware incident representation. In this RQ, we examine the effectiveness of hierarchy-guided contrastive learning on the fault pattern profiling task. We first compare the performance of FaultProfIT against its variants by removing or replacing different components in FaultProfIT. Then, we analyze the label representations of fault patterns to reveal the learnt hierarchy. Specially, we consider the following variants and report the precision, recall, and F1 score on level IV for illustration.

- *r.p.* GCN: We replace Graphormer with Graph Convolutional Network [32] (GCN) as the backbone of hierarchy encoder.
- *r.p.* GAT: We replace Graphormer with Graph Attention Network [49] (GAT) to encode hierarchy.
- *w.o.* Graphormer: We remove then Graphormer encoder and directly use the node vectors to guide positive sample construction.
- *w.o.* description embedding: We remove description embedded in the hierarchy encoder and only use the label embedding.
- *w.o.* contrastive loss: We remove contrastive loss function.
- *w.o.* augmented samples loss: We remove the classification loss function for augmented positive samples.

**Table 3: Experiment results of different models.**

| Method | Precision | Recall | F1-score |
|---|---|---|---|
| **FaultProfIT** | **76.6** | **80.1** | **78.3** |
| *-r.p.* GCN | 71.4 | 74.2 | 72.8 |
| *-r.p.* GAT | 71.9 | 74.8 | 73.3 |
| *-w.o.* description embedding | 72.8 | 75.1 | 74.0 |
| *-w.o.* Graphormer | 66.2 | 71.8 | 68.9 |
| *-w.o.* contrastive loss | 67.2 | 75.5 | 71.3 |
| *-w.o.* augmented samples loss | 53.4 | 64.4 | 58.4 |
| *-w.o.* whole contrastive module | 50.6 | 59.5 | 54.7 |

- *w.o.* whole contrastive module: We remove the entire hierarchy-guided contrastive learning module and only leverage the incident encoder and classification module for profiling.

**Results.** The results in Table 3 shows that: (1) Graphormer exhibits superior performance compared to both GCN and GAT hierarchy encoders in this task. This can be attributed to the fact that GCN and GAT encode local structures, which only perform convolutions or attentions on neighboring nodes, whereas Graphormer employs global attention, allowing each node to attend to all others in the graph. This global attention mechanism is more effective in encoding hierarchy. (2) Removing the description embedding from the node embedding leads to a decrease in performance for FaultProfIT. This result highlights the significance of fault descriptions, as they provide additional semantic information that helps in learning more effective representations. (3) When the Graphormer encoder is entirely discarded, there is a significant drop in performance, indicating the usefulness of hierarchy in providing organized label relations that guide fault pattern profiling. (4) Without the training objectives of contrastive loss or classification loss of positive samples, FaultProfIT performs poorly. This indicates that both the positive sample construction and the contrastive learning framework contribute to FaultProfIT. The positive samples are useful even without contrastive learning, and contrastive learning can further enhance the model by constraining the vector space. (5) Removing the entire hierarchy-guided contrastive learning module results in a substantial accuracy decline for FaultProfIT. This finding again confirms the effectiveness of our method in hierarchical fault pattern profiling.

**Visualization.** To investigate the encoding of the taxonomic hierarchy, we visually analyze the distributions of fault pattern embeddings. Specifically, we consider the weight matrix $W_c \in \mathbb{R}^{k \times d_h}$ in Equation 6 as fault pattern representations. where each row represents a node in the taxonomy. We employ the T-SNE algorithm [47] with default parameters to project the high-dimensional vectors into a two-dimensional space. The resulting points are plotted in Figure 7, where points with the same color correspond to fault patterns from the same parent node. For comparison, we also visualize the embeddings produced by MacBERT. Our observations reveal that the fault pattern embeddings of MacBERT are scattered, while the embeddings of FaultProfIT exhibit clustering based on the parent nodes. This behavior arises from the fact that the representation of a label and its parent is trained to be similar, as they are classified simultaneously. Consequently, if the hierarchy is incorporated into the text representation, labels sharing the same parent should possess more similar representations compared to those with different parents.

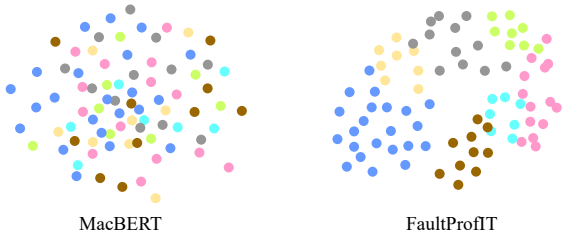


**Figure 7: Visualization of the fault pattern representations.**

## 4.4 RQ3: How does FaultProfIT perform on diverse types of incidents?

**Setup.** Developing a strong fault pattern profiler necessitates a substantial volume of training data. To accommodate this requirement, we utilize a number of historical incident tickets collected from a variety of services and encompassing a range of severity levels. In this RQ, we explore the performance of FaultProfIT across diverse incident types. To this end, we partition the testset into several subsets based on their severity and the services they impact. Given the existence of over 30 distinct services, we categorize them into five primary groups: *Infrastructure*, *Computing*, *Networking*, *Storage*, and *Others*.

**Results.** Figure 8 shows the F1-score of FaultProfIT across various incident types. From the left segment of Figure 8, we can find that FaultProfIT exhibits superior performance on incidents with less severity, especially those of S5 level, compared to more severe ones. This could be attributed to the increased complexity in diagnosing severe incidents, which often involve extended incident contexts and a greater number of affected components, thereby posing a greater challenge for FaultProfIT to identify. These findings further confirm the viability of automated fault pattern profiling within cloud systems, given the higher frequency of less severe incident tickets and fewer human resources allocated for analysing these tickets, compared to those of higher severity.

The right segment of Figure 8 indicates that the accuracy varies across services. Incidents impacting services within the *Infrastructure* and *Computing* categories yield a relatively high F1-score. Conversely, incidents affecting services within *Storage* or *Others* categories demonstrate lower accuracy. Different from other services, the *Infrastructure* and *Computing* services mainly experience faults within clusters and hosts, which comprise servers and hardware. These incidents often exhibit explicit descriptive signals such as "server" or "datacenter", thereby facilitating easier classification. The lower F1-score associated with *Storage* services can be attributed to the smaller number of incidents within this category present in our dataset, as the service is more robust and produces fewer failures. The low accuracy of the *Others* category is because it consists of multiple services, each with diverse phenomenons and mitigation methods. As a result, FaultProfIT encounters difficulties in discerning common semantic patterns within this category, leading to a higher rate of erroneous predictions.

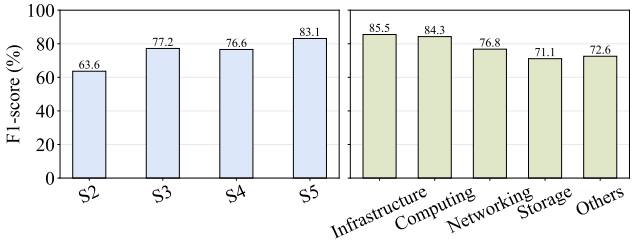


**Figure 8: Results of FaultProfIT for incidents of different severities and services.**

## 5 DEPLOYMENT EXPERIENCE

In this section, we share our experience of deploying FaultProfIT in product X, a cloud reliability analysis system at CloudA. This system provides an extensive array of centralized analytic functionalities for incident management, including tracing, retrieval, analysis, and modeling. These functionalities are tailored to support a variety of service teams and engineers in managing their production incidents and identifying reliability issues. At CloudA, the reliability team conducts incident postmortem analysis with product X. One of the major outcomes is the profiling of fault patterns, which are subsequently utilized for trend analysis and vulnerability identification. In most cases, the reliability team does not conduct postmortem analysis once the incident is mitigated. Instead, they perform analysis periodically, such as weekly or monthly, selecting a set of severe incidents to investigate and profile fault patterns.

Traditionally, the process of fault pattern profiling relied on manual labeling. However, with the introduction of new service products and the increasing number of customers, engineers found it increasingly challenging to analyze emerging incidents. To reduce efforts, reliability engineers prioritized severe incidents (*i.e.*, S1, S2, and S3) for profiling and proposed product improvement suggestions based on the profiling results. However, such practices neglect the incidents with less severity. Even though system updates were frequently released to improve reliability, the number of minor incidents continued to increase. Although such incidents did not cause severe impacts thanks to the fault tolerance measures, specific customers suffered from occasional performance degradation or network interruption, affecting customer experience and causing complaints. Therefore, the integration of automated tools for fault pattern profiling is essential to improve both the efficiency and comprehensiveness of the analysis process.

To achieve this goal, we have integrated FaultProfIT into product X. Currently, *10,000+* of incidents from *30+* cloud services (including historical incidents) have been analyzed by FaultProfIT for fault pattern profiling over a *six-month* period. Concretely, we provide an API interface in Product X. Engineers can invoke the API to call FaultProfIT, which automatically analyzes incident tickets to profile fault patterns. When the API is invoked, the unanalyzed tickets are sent to the server. Subsequently, FaultProfIT conducts data preprocessing and predicts the labels in a batch manner. Once the prediction is completed, the profiled fault patterns are returned and visualized on the frontend of product X.

To show how the predicted fault patterns are utilized in CloudA, we collected incidents created over the most recent 26 weeks after the deployment. Their fault patterns are automatically profiled by

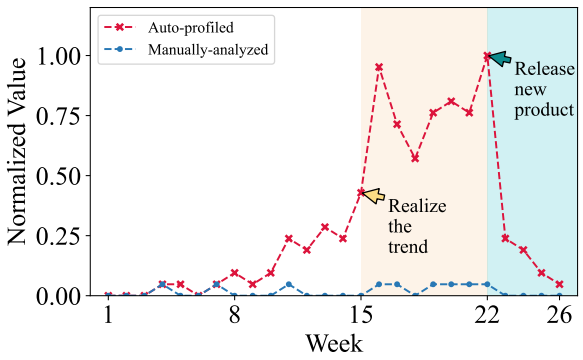


**Figure 9: An example of trends analysis for the *memory overload* fault pattern with FaultProfIT.**

FaultProfIT on a weekly basis. We present an example trend of the fault pattern named *memory overload* from a computing service, which indicates that the system's memory usage had exceeded its capacity. Figure 9 shows the trends of incidents with memory overload from two resources[2]. The blue line represents manually analyzed postmortem reports, which are conducted only on severe incidents, and the red line represents automatically profiled incidents. In the first 10 weeks, the occurrence of incidents of the fault pattern remained at a low level. Then from week 10 onwards, the number of auto-profiled incidents began to rise, reaching a peak at week 16. Without the integration of FaultProfIT, engineers would be unaware of the faults their service was undergoing, as the memory overload is not a severe fault to cause large impacts and thus they are not analyzed in the postmortem. However, with FaultProfIT, they were alerted to an increasing number of memory overload incidents at week 15. Therefore, the team initiated a set of actions to investigate the overload issues within the service. During the two-month investigation, engineers conducted numerous experiments to test the system, identify weaknesses, and fix the defects. Finally, they released a new version of the service at week 22, and this fault pattern began to decrease and eventually returned to a low level by week 26.

## 6 RELATED WORKS

With years of efforts, researchers have conducted numerous studies [14, 24, 26, 37] and proposed many automatic approaches [15, 22, 23, 33, 33–35, 58] on cloud incidents management. Among these works, Gunawi et al. [24] discussed why incidents still take place in cloud systems by analyzing public incident reports of popular cloud services. Chen et al. [14] presented a comprehensive study on how incidents are managed in Microsoft Azure.

Timely and accurate incident detection can facilitate the quick response of engineers, accelerating the procedures involved in incident management. By analyzing cloud system service behaviors, Warden [34] was proposed to analyze system-wide alerting signals from a global view for proactive incident detection. To avoid the flooding issue reports, MID [22] was proposed to identify incidents from large-amount, multi-dimensional issue reports.

[2]Due to confidential reasons, we present a normalized occurrence number to reflect the trends.

Once the incidents are detected, diagnosis and Root Cause Analysis (RCA) are conducted to obtain comprehensive information that aids in follow-up triage and choosing effective mitigation strategies. Onion [58] localizes the incident-indicating logs from the incident context, where a contrast analysis is then performed to accurately find out a few lines of root cause related log. ESRO [8] constructs a unified graph of alerts and incident reports to recommend root causes and remediation steps. iPACK [38] and LinkCM [23] were proposed to link and aggregate duplicate incidents by fusing the failure information between the customer-reported tickets and the machine-generated incidents. To avoid excessive aggregation of incidents, HALO [57] was proposed to localize the fault to a proper granularity, which usually suffers from improper aggregation level of incidents for further diagnosis and triage. Chen et al. [9] conducted an empirical study about incident triage in Microsoft Azure and further proposed DeepCT [10] to automate continuous incident triage for further incident mitigation.

After the mitigation of incidents, postmortem analyses are conducted to provide fruitful insights and experiences. This knowledge can assist in profiling the fault, contributing significantly to the enhancement of the system's stability and response speed. Shetty et al. [46] conducted an empirical study on hundreds of high severity incidents postmortems in a large-scale cloud service and provided guidance on how to tackle future incidents. AutoARTS [18] was proposed to label incident root causes by analyzing potential contributing factors with knowledge gained from incident postmortems.

The rise of Large Language Model (LLM) has brought new opportunities to the field of intelligent incident management. With intrinsic domain knowledge, LLM can diagnose and interpret incidents like on-call engineers (OCEs). Ahmed et al. [6] effectively fine-tuned LLMs to suggest the root cause and mitigation strategies for cloud incidents, combining both external domain expertise and internal pre-trained model knowledge. Moreover, RCACopilot [12] was proposed to summarize the incidents and predict the incident's root cause with generated explanations by employing LLMs.

Our work focuses on the postmortem analysis phase of incident management. We distinguish from existing works by developing an automated approach to profile fault patterns based on incident tickets, which can handle emerging and less severe incidents. Our work can provide a comprehensive view of a range of incidents and improve the efficiency of reliability engineers.

## 7 CONCLUSION

Fault pattern profiling is an important task of incident postmortem analysis in large-scale cloud systems. To support consistent and large-scale fault pattern profiling, we introduce FaultProfIT, an automated approach that leverages hierarchical text classification. FaultProfIT takes the textual incident context as input and applies language models to predict fault pattern labels. To inject hierarchy information into the taxonomy and mitigate the data insufficiency problem, we employ hierarchy-guided contrastive learning to enhance the incident representations. We evaluate our approach to the production incidents of CloudA, a top-tier global cloud provider. Experimental results demonstrate the high F1-score achieved by FaultProfIT and the effectiveness of hierarchy-guided contrastive learning. Furthermore, we have deployed FaultProfIT at the reliability analysis platform of CloudA for a duration of six months, gaining valuable insights and experience from the deployment.

## ACKNOWLEDGEMENT

We sincerely thank the anonymous reviewers for their constructive comments and suggestions. The work described in this paper was supported by the Research Grants Council of the Hong Kong Special Administrative Region, China (No. CUHK 14206921 of the General Research Fund).

## REFERENCES

[1] 2021. 2021 Facebook outage. https://en.wikipedia.org/wiki/2021_Facebook_outage. [Online; accessed 31 July 2023].
[2] 2023. AWS Post-Event Summaries. https://aws.amazon.com/cn/premiumsupport/technology/pes/. [Online; accessed 31 July 2023].
[3] 2023. Azure status history. https://azure.status.microsoft/en-us/status/history/. [Online; accessed 31 July 2023].
[4] 2023. Google Cloud Status Dashboard. https://status.cloud.google.com/summary. [Online; accessed 31 July 2023].
[5] Salman Ahmed, Muskaan Singh, Brendan Doherty, Effirul Ramlan, Kathryn Harkin, Magda Bucholc, and Damien Coyle. 2023. Knowledge-based intelligent system for IT incident DevOps. In *2023 IEEE/ACM International Workshop on Cloud Intelligence & AIOps (AIOps)*. IEEE, 1–7.
[6] Toufique Ahmed, Supriyo Ghosh, Chetan Bansal, Thomas Zimmermann, Xuchao Zhang, and Saravan Rajmohan. 2023. Recommending Root-Cause and Mitigation Steps for Cloud Incidents Using Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1737–1749.
[7] Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. 2018. Generating Natural Language Adversarial Examples. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. 2890–2896.
[8] Sarthak Chakraborty, Shubham Agarwal, Shaddy Garg, Abhimanyu Sethia, Udit Narayan Pandey, Videh Aggarwal, and Shiv Saini. 2023. ESRO: Experience Assisted Service Reliability against Outages. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 255–267.
[9] 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)*. IEEE, 111–120.
[10] 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)*. IEEE, 364–375.
[11] Ting Chen, Simon Kornblith, Mohammad Norouzi, and Geoffrey Hinton. 2020. A simple framework for contrastive learning of visual representations. In *International conference on machine learning*. PMLR, 1597–1607.
[12] Yinfang Chen, Huaibing Xie, Minghua Ma, Yu Kang, Xin Gao, Liu Shi, Yunjie Cao, Xuedong Gao, Hao Fan, Ming Wen, et al. 2023. Empowering Practical Root Cause Analysis by Large Language Models for Cloud Incidents. *arXiv preprint arXiv:2305.15778* (2023).
[13] Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, et al. 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*. 304–314.
[14] Zhuangbin Chen, Yu Kang, Liqun Li, Xu Zhang, Hongyu Zhang, Hui Xu, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Towards intelligent incident management: why we need it and how we make it. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1487–1497.
[15] Zhuangbin Chen, Jinyang Liu, Yuxin Su, Hongyu Zhang, Xuemin Wen, Xiao Ling, Yongqiang Yang, and Michael R Lyu. 2021. Graph-based incident aggregation for large-scale online service systems. In *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 430–442.
[16] Yiming Cui, Wanxiang Che, Ting Liu, Bing Qin, Shijin Wang, and Guoping Hu. 2020. Revisiting Pre-Trained Models for Chinese Natural Language Processing. In *Findings of the Association for Computational Linguistics: EMNLP 2020*. 657–668.
[17] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. 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-NLT)*. 4171–4186.

[18] Pradeep Dogga, Chetan Bansal, Richard Costleigh, Gopinath Jayagopal, Suman Nath, and Xuchao Zhang. 2023. AutoARTS: Taxonomy, Insights and Tools for Root Cause Labelling of Incidents in Microsoft Azure. In *2023 USENIX Annual Technical Conference (USENIX ATC 23)*. 359–372.

[19] Zhengxiao Du, Yujie Qian, Xiao Liu, Ming Ding, Jiezhong Qiu, Zhilin Yang, and Jie Tang. 2022. GLM: General Language Model Pretraining with Autoregressive Blank Infilling. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 320–335.

[20] Jiaqi Gao, Nofel Yaseen, Robert MacDavid, Felipe Vieira Frujeri, Vincent Liu, Ricardo Bianchini, Ramaswamy Aditya, Xiaohang Wang, Henry Lee, David Maltz, et al. 2020. Scouts: Improving the diagnosis process through domain-customized incident routing. In *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication*. 253–269.

[21] Supriyo Ghosh, Manish Shetty, Chetan Bansal, and Suman Nath. 2022. How to fight production incidents? an empirical study on a large-scale cloud service. In *Proceedings of the 13th Symposium on Cloud Computing*. 126–141.

[22] Jiazhen Gu, Chuan Luo, Si Qin, Bo Qiao, Qingwei Lin, Hongyu Zhang, Ze Li, Yingnong Dang, Shaowei Cai, Wei Wu, et al. 2020. Efficient incident identification from multi-dimensional issue reports via meta-heuristic search. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 292–303.

[23] Jiazhen Gu, Jiaqi Wen, Zijian Wang, Pu Zhao, Chuan Luo, Yu Kang, Yangfan Zhou, Li Yang, Jeffrey Sun, Zhangwei Xu, et al. 2020. Efficient customer incident triage via linking with system incidents. In *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 1296–1307.

[24] Haryadi S Gunawi, Mingzhe Hao, Riza O Suminto, Agung Laksono, Anang D Satria, Jeffry Adityatama, and Kurnia J Eliazar. 2016. Why does the cloud stop computing? lessons from hundreds of service outages. In *Proceedings of the Seventh ACM Symposium on Cloud Computing (SOCC)*. 1–16.

[25] Junjie Huang, Duyu Tang, Linjun Shou, Ming Gong, Ke Xu, Daxin Jiang, Ming Zhou, and Nan Duan. 2021. CoSQA: 20,000+ Web Queries for Code Search and Question Answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 5690–5700.

[26] Peng Huang, Chuanxiong Guo, Lidong Zhou, Jacob R Lorch, Yingnong Dang, Murali Chintalapati, and Randolph Yao. 2017. Gray failure: The achilles' heel of cloud-scale systems. In *Proceedings of the 16th Workshop on Hot Topics in Operating Systems (HotOS)*. 150–155.

[27] Eric Jang, Shixiang Gu, and Ben Poole. 2016. Categorical Reparameterization with Gumbel-Softmax. In *International Conference on Learning Representations*.

[28] Pengxiang Jin, Shenglin Zhang, Minghua Ma, Haozhe Li, Yu Kang, Liqun Li, Yudong Liu, Bo Qiao, Chaoyun Zhang, Pu Zhao, et al. 2023. Assess and Summarize: Improve Outage Understanding with Large Language Models. *arXiv preprint arXiv:2305.18084* (2023).

[29] Vladimir Karpukhin, Barlas Oguz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. 2020. Dense Passage Retrieval for Open-Domain Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*. 6769–6781.

[30] Taeuk Kim, Kang Min Yoo, and Sang-goo Lee. 2021. Self-Guided Contrastive Learning for BERT Sentence Representations. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*. 2528–2540.

[31] Diederik P Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980* (2014).

[32] Thomas N Kipf and Max Welling. 2016. Semi-Supervised Classification with Graph Convolutional Networks. In *International Conference on Learning Representations*.

[33] Cheryl Lee, Tianyi Yang, Zhuangbin Chen, Yuxin Su, and Michael R Lyu. 2023. Maat: Performance Metric Anomaly Anticipation for Cloud Services with Conditional Diffusion. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 116–128.

[34] Liqun Li, Xu Zhang, Xin Zhao, Hongyu Zhang, Yu Kang, Pu Zhao, Bo Qiao, Shilin He, Pochian Lee, Jeffrey Sun, Feng Gao, Li Yang, Qingwei Lin, Saravanakumar Rajmohan, Zhangwei Xu, and Dongmei Zhang. 2021. Fighting the Fog of War: Automated Incident Detection for Cloud Systems. In *2021 USENIX Annual Technical Conference (USENIX ATC)*. USENIX Association, 131–146. https://www.usenix.org/conference/atc21/presentation/li-liqun

[35] Yichen Li, Xu Zhang, Shilin He, Zhuangbin Chen, Yu Kang, Jinyang Liu, Liqun Li, Yingnong Dang, Feng Gao, Zhangwei Xu, et al. 2022. An Intelligent Framework for Timely, Accurate, and Comprehensive Cloud Incident Detection. *ACM SIGOPS Operating Systems Review* 56, 1 (2022), 1–7.

[36] Zeyan Li, Nengwen Zhao, Mingjie Li, Xianglin Lu, Lixin Wang, Dongdong Chang, Xiaohui Nie, Li Cao, Wenchi Zhang, Kaixin Sui, et al. 2022. Actionable and interpretable fault localization for recurring failures in online service systems. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 996–1008.

[37] Haopeng Liu, Shan Lu, Madan Musuvathi, and Suman Nath. 2019. What bugs cause production cloud incidents?. In *Proceedings of the Workshop on Hot Topics in Operating Systems (HotOS)*. 155–162.

[38] Jinyang Liu, Shilin He, Zhuangbin Chen, Liqun Li, Yu Kang, Xu Zhang, Pinjia He, Hongyu Zhang, Qingwei Lin, Zhangwei Xu, Saravan Rajmohan, Dongmei Zhang, and Michael R. Lyu. 2023. Incident-Aware Duplicate Ticket Aggregation for Cloud Systems. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. 2299–2311.

[39] Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *Comput. Surveys* 55, 9 (2023), 1–35.

[40] Xiao Liu, Kaixuan Ji, Yicheng Fu, Weng Tam, Zhengxiao Du, Zhilin Yang, and Jie Tang. 2022. P-tuning: Prompt tuning can be comparable to fine-tuning across scales and tasks. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*. 61–68.

[41] Jian-Guang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: An experience report. In *2013 28th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 475–485.

[42] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems (NeurIPS)* 32 (2019).

[43] Jiaxing Qi, Zhongzhi Luan, Shaohan Huang, Carol Fung, Hailong Yang, Hanlu Li, Danfeng Zhu, and Depei Qian. 2023. LogEncoder: Log-based Contrastive Representation Learning for anomaly detection. *IEEE Transactions on Network and Service Management* (2023).

[44] Nils Rethmeier and Isabelle Augenstein. 2023. A Primer on Contrastive Pretraining in Language Processing: Methods, Lessons Learned, and Perspectives. *Comput. Surveys* 55, 10 (2023), 1–17.

[45] GM Shahariar, Tahmid Hasan, Anindya Iqbal, and Gias Uddin. 2023. Contrastive Learning for API Aspect Analysis. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 637–648.

[46] Manish Shetty, Chetan Bansal, Sumit Kumar, Nikitha Rao, and Nachiappan Nagappan. 2022. SoftNER: Mining knowledge graphs from cloud incidents. *Empirical Software Engineering* 27, 4 (2022), 93.

[47] Laurens Van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, 11 (2008).

[48] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[49] Petar Veličković, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Liò, and Yoshua Bengio. 2018. Graph Attention Networks. In *International Conference on Learning Representations*.

[50] Weijing Wang, Junjie Chen, Lin Yang, Hongyu Zhang, Pu Zhao, Bo Qiao, Yu Kang, Qingwei Lin, Saravanakumar Rajmohan, Feng Gao, et al. 2021. How long will it take to mitigate this incident for online service systems?. In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 36–46.

[51] Zihan Wang, Peiyi Wang, Lianzhe Huang, Xin Sun, and Houfeng Wang. 2022. Incorporating Hierarchy into Text Encoder: a Contrastive Learning Approach for Hierarchical Text Classification. In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 7109–7119.

[52] Moshi Wei, Nima Shiri Harzevili, Yuchao Huang, Junjie Wang, and Song Wang. 2022. Clear: contrastive learning for api recommendation. In *Proceedings of the 44th International Conference on Software Engineering*. 376–387.

[53] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, et al. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv preprint arXiv:1910.03771* (2019).

[54] Zhuofeng Wu, Sinong Wang, Jiatao Gu, Madian Khabsa, Fei Sun, and Hao Ma. 2020. Clear: Contrastive learning for sentence representation. *arXiv preprint arXiv:2012.15466* (2020).

[55] Chengxuan Ying, Tianle Cai, Shengjie Luo, Shuxin Zheng, Guolin Ke, Di He, Yanming Shen, and Tie-Yan Liu. 2021. Do transformers really perform badly for graph representation? *Advances in Neural Information Processing Systems* 34 (2021), 28877–28888.

[56] Alessandro Zangari, Matteo Marcuzzo, Michele Schiavinato, Matteo Rizzo, Andrea Gasparetto, Andrea Albarelli, et al. 2023. Hierarchical Text Classification: a review of current research. *EXPERT SYSTEMS WITH APPLICATIONS* 224 (2023).

[57] Xu Zhang, Chao Du, Yifan Li, Yong Xu, Hongyu Zhang, Si Qin, Ze Li, Qingwei Lin, Yingnong Dang, Andrew Zhou, et al. 2021. Halo: Hierarchy-aware fault localization for cloud systems. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*. 3948–3958.

[58] Xu Zhang, Yong Xu, Si Qin, Shilin He, Bo Qiao, Ze Li, Hongyu Zhang, Xukun Li, Yingnong Dang, Qingwei Lin, et al. 2021. Onion: identifying incident-indicating logs for cloud systems. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE)*. 1253–1263.

[59] Yujin Zhao, Ling Jiang, Ye Tao, Songlin Zhang, Changlong Wu, Yifan Wu, Tong Jia, Ying Li, and Zhonghai Wu. 2023. How to Manage Change-Induced Incidents? Lessons from the Study of Incident Life Cycle. In *2023 IEEE 34th International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 264–274.

[60] Jie Zhou, Chunping Ma, Dingkun Long, Guangwei Xu, Ning Ding, Haoyu Zhang, Pengjun Xie, and Gongshen Liu. 2020. Hierarchy-aware global model for hierarchical text classification. In *Proceedings of the 58th annual meeting of the association for computational linguistics*. 1106–1117.