LogKG: Robust Log Failure Diagnosis through Knowledge Graph

Yicheng Sui, Yuzhe Zhang, Jianjun Sun, Ting Xu, Shenglin Zhang, *Member, IEEE,* Zhengdan Li, Yongqian Sun, *Member, IEEE,* Fangrui Guo, Junyu Shen, Yuzhi Zhang, Dan Pei, *Senior Member, IEEE,* Li Yu

***Abstract*—Logs are one of the most valuable data to describe the running state of services. Failure diagnosis through logs is crucial for service maintenance. The current automatic log failure diagnosis methods focus on the semantics of log templates. They do not fully use the multi-field information in the logs, which fails to capture the relationship between them. In this paper, we propose LogKG, a new framework for diagnosing failures based on automatic knowledge graphs of logs. LogKG fully extracts entities and relationships from logs to mine multi-field information and their relationships through knowledge graphs. To fully use the information represented by the knowledge graph, we propose a failure-oriented log representation (FLOR) method to extract failure-related representation vectors from failure cases. Utilizing the OPTICS clustering method, LogKG aggregates historical failure cases, labels typical failure cases, and trains a failure diagnosis model to identify the root cause. We evaluate the effectiveness of LogKG on a real-world log dataset and a public log dataset, showing that it outperforms existing methods. With the deployment in a top-tier global Internet Service Provider (ISP), we demonstrate the performance and practicability of LogKG.**

***Index Terms*—LogKG, cluster,embedding, diagnosis.**

Component	*Task ID*	*Content*
mobservice2	1ce0358e241dc150	now call service:redisservice2
mobservice1	5c85bd2cf59342ec	now call service:redisservice2
redisservice2	1ce0358e241dc150	QR code has expired
redisservice1	5c85bd2cf59342ec	redis write success
redisservice1	5c85bd2cf59342ec	redis read information successfully
redisservice1	5c85bd2cf59342ec	service accept
mobservice2	1ce0358e241dc150	information has expired
mobservice1	5c85bd2cf59342ec	info write cussess
webservice2	1ce0358e241dc150	an error occurred in the downstream service
webservice1	5c85bd2cf59342ec	write redis successfully

Component	*Task ID*	*Content*
dbservice2	efccf0c61a3fd9ce	now call service:redisservice1
dbservice1	d04ac67cf6c285cd	now call service:redisservice1
redisservice1	d04ac67cf6c285cd	service refuse
redisservice1	efccf0c61a3fd9ce	redis write success
dbservice1	d04ac67cf6c285cd	dbservice1 access redis service denied
dbservice2	efccf0c61a3fd9ce	token generate success
webservice2	d04ac67cf6c285cd	an error occurred in the downstream service
webservice1	efccf0c61a3fd9ce	write redis successfully

Fig. 1: Logs of two failure cases.

I. Introduction

LARGE-SCALE services are being developed and implemented at an increasing rate, resulting in increased complexity and interdependence. As a result, when one service fails, several other services may also suffer, affecting the experiences of millions of users [1] [2]. An accurate and practical failure diagnosis approach can considerably improve service security and reliability. Log-based failure diagnostic approaches rely on logs to establish the root cause of a failure [3]. Such as message queue service failure, database service failure, computing node unavailable, virtual machine creation failure, etc. Since logs are often the only accessible data for capturing service runtime information, log-based failure diagnostic approaches have attracted much attention recently [4].

Logs contain rich semantic information about service systems, such as the system's operation status, which is vitally essential for system maintenance.They are usually semi-structured text generated by logging statements in source code. Table I lists several examples of logs in the Generic AIOps Atlas (GAIA) dataset[1]. A log usually has multiple fields, including timestamp, level, IP, component, task ID, and content, which represent when it is generated, the severity of the event, the device, software component, and process that generate it, and the detailed event information, respectively. Comprehensively considering these different fields' information makes it more effective to mine the failure-related logs. For example, Figure 1 shows the logs during two failure cases. The logs highlighted in red indicate the root causes of these failures. Finding them from the raw logs using simply the content field is challenging. When the information of other fields is considered holistically, things get easier. The failure-related logs for the first case in Figure 1 can be easily captured using the task ID "1ce0358e241dc150" and the components "redisservice2" and "mobservice2", combined with the semantic information in the content field. For the second case, the task ID "d04ac67cf6c285cd", and the components "redisservice1" and "dbservice1" make it simple to discover them. Further, they can be used to determine the failures' root cause.

Some log-based failure diagnosis methods [3], [5]–[9] mine the information contained in the log context. However, the preceding studies only focus on the unstructured content field

S. Zhang is the corresponding author. Y. Sui, Z. Zhang, S. Zhang, Y. Sun, F. Guo, J. Shen and Y. Zhang are with Nankai University, Tianjin, China.

D. Pei is with Department of Computer Science, Tsinghua University, Beijing, China, and also with Beijing National Research Center for Information Science and Technology. Email: peidan@tsinghua.edu.cn.

[1]GAIA is an overall dataset for analyzing operation problems such as anomaly detection, log analysis, failure localization, etc. https://github.com/CloudWise-OpenSource/GAIA-DataSet

TABLE I: Emamples of Logs

Timestamp	Level	IP	Component	Task ID	Content
2021-07-04 00:38:16,368	INFO	0.0.0.2	logservice2	6318eaeaabe5ee2b	the list of all available services are dbservice1: http://0.0.0.4:9388, dbservice2: http://0.0.0.2:9389
2021-07-04 00:42:20,756	INFO	0.0.0.2	dbservice2	a80b24eb9b65be4c	now call service:redisservice2, inst: http://0.0.0.2:9387 as a downstream service
2021-07-04 00:42:45,260	WARNING	0.0.0.1	redisservice1	a39e68cffda53b88	User not scanned, please wait. Status_code: 300
2021-07-04 00:46:59,157	INFO	0.0.0.1	mobservice1	dd29a2b8bd9d9766	info 487a2bca-dc1e-11eb-b1b8-0242ac110004: XWkAGNLi write success

in the logs, neglecting the information of the structured fields, making them unable to mine the failures' features fully. Existing studies [5], [6]have shown that knowledge graphs can effectively fuse unstructured texts and structured data. Therefore, we fuse unstructured content field and structured fields (e.g., timestamp, level, IP, component, task ID) in the logs based on a knowledge graph and utilize it for failure diagnosis. A knowledge graph uses entities and relationships to mine the information in the fields comprehensively.

Existing KG-based text fusion methods [6] usually consist of three steps: *entity extraction, relation extraction, and entity alignment*. Entities extracted from logs are associated with log entities, and log entities are associated with template entities. Therefore, the knowledge graph constructed based on logs only has several pre-defined types of relationships which do not need to be extracted. As a result, the primary task of multi-field information fusion for logs is to extract entities from various fields and align them based on semantics. The information contained in the unstructured content field, as listed in Table I, is relatively fixed and straightforward. We can easily extract the entities from them. For the unstructured texts, the existing log parsing methods [10], [11] can effectively resolve the texts into templates and parameters, which can be used to extract and align the entities. Then, entities with different representations referring to the same object are aggregated into entities in the knowledge graph. In this way, entities and relationships included in logs can be retrieved, and We can construct a knowledge graph to fuse the multi-field information.

To overcome the challenges lying in entity extraction, entity alignment, and failure case representation (more details can be seen in section §II.c), we propose LogKG, a novel framework for failure diagnosis based on KG-based multi-field information fusion. LogKG extracts different types of entities and relationships from multi-field in logs and aligns the entities based on semantics. It represents logs utilizing knowledge graph embedding and obtains failure-related representation vectors from failure cases. Then it aggregates historical failure cases, labels typical failure cases, and trains a failure diagnosis model to identify the type of the new failure. Above all, the following are the primary contributions of this paper:

1) **Log event extraction.** To extract entities from the multiple fields of logs, we propose a novel log event extraction method, which performs open information extraction. It extracts triples from log templates and treats them as independent events. These events provide a more comprehensive representation of logs and can be used in knowledge graph construction.
2) **Alignment for triples and parameters.** To align the entities extracted from the multiple fields of logs, we propose an entity alignment method, which is effective for aggregating the entities referring to the same object and establishing indirect associations between them. To align the parameters of the content field, We propose a context-based method for identifying parameter types. To align the log triples, we propose a semantics-based clustering method, which generates BERT variables to obtain vectors of these triples, and uses a clustering method to aggregate the triples into event entities.
3) **Failure-Oriented Log Representations.** To effectively mine failure features, we propose a failure-oriented log representation (FOLR) method to extract failure representation vectors from failure-related logs. FOLR finds failure-related logs after the failure occurs and then calculates the failure representation vector.

To measure the effectiveness of LogKG, we evaluate it on a real-world dataset collected from china Mobile (CMCC) , atop-tier global Internet Service Provider (ISP), and a popular open-source dataset from GAIA. LogKG improvs accuracy and Macro-F1 scores by 4% and 1.5%, respectively, over the two baselines on the CMCC dataset. At the same time, the improvement on the GAIA dataset is 26% and 15%, respectively. We also demonstrate the effectiveness of FOLR and knowledge graph through ablation studies.

Through the case study, we evaluate the workload of manual verification and automatic failure diagnosis based on LogKG. Operators need to check 2500+ logs to find the logs indicating the root cause, which requires much manual work. However, LogKG handles an average of 47 cloud-based network failures per day, which can be classified and located in less than 5 minutes, reducing the average failure repair time by more than 20 minutes. LogKG significantly reduces the amount of manual work required.

The remainder of this paper is organized as follows: We will introduce the background of our work in §I. We introduce the related works in §II. In §III, we will introduce LogKG frame-

work. §IV introduces our experimental setup and results. §V describes the actual case where we use LogKG for knowledge graph construction and failure diagnosis, followed by a paper summary in §VI.

II. Related Work and Preliminaries

We first present representative approaches related to semantic information extraction and failure diagnosis of logs. Then, we give the natural language processing (NLP) concepts used in this paper. Finally, we present three challenges we face.

A. The work of related representative methods

1) Log-based Semantic Information Extraction: Some existing log anomaly detection works [1], [12]–[15] have proposed to extract the semantic information of logs. LogAnomaly [1] obtains the semantic vector of the template by determining synonyms and antonyms to generate template vectors. LogRobust [12] uses TF-IDF to give different word vectors with different weights to obtain template vectors. [13] combines the transaction-level topic modeling for learning the embedding of logs, where a transaction is a group of logs sequentially occurring in a time window. Log2vec [14] uses the semantic association to represent the relationship between logs by constructing heterogeneous graphs and obtains the vector representation of logs by a graph embedding-based method.

Different from the above methods extracting the semantic information in logs from the level of log entries, SLOGERT [16] and LEKG [15] model semantic information in logs based on knowledge graph. SLOGERT [16] obtains keywords information contained in the log templates through CoreNLP [17]. In addition, it uses regular expressions to obtain parameter type information. LEKG [15] uses extracted semantic information to construct triples through NER and background knowledge, graph and generates new triples through rule inference. The log-based knowledge graph is combined with the background knowledge graph, making better use of the vast amount of information contained in background knowledge. None of these methods mines the semantics of the unstructured fields of logs and fuses the unstructured data.

2) Log-based Failure Diagnosis: Manual failure diagnosis is often error-prone and labor-intensive [4].Therefore, some methods have been proposed for automatic failure diagnosis based on logs in recent years [20] [3] [19] [22] [27] [9] [24] [28] [26] [25]. Table II lists some related works of failure diagnosis and why they are chosen or not as baselines in our work. We analyze the data types in the logs to classify them into the following five categories.

a) DeepLog and LogFlash. DeepLog [20] and LogFlash [19] use logs to construct the log execution path and diagnose the root cause of failuresAfter a service fails, they mine the causes of the failure through log execution path. Since the above two failed to perform path mining through logs, it is inapplicable to our circumstances.

b) LOGAN, Onion, and FDiagV3. The fundamental idea behind all the three methods is to extract components from logs, compare normal and abnormal logs, and select the logs indicating failure information to aid in failure diagnosis. LOGAN [9] captures normal log patterns by log grouping, log parsing, and log alignment. When a failure occurs, it calculates the divergence of the log sequence from a reference model and suggests the possible root cause. As an aided failure diagnosis method, Onion [7] extracts the semantic information in the log, compares the normal and anomalous logs, and selects the log that indicates the failure root cause. Similarly, FDiagV3 [8] mines failure-related logs from computer clusters to aid in failure diagnosis. Because the target of the three methods is finding logs that potentially indicate failure root cause, which is inconsistent with our objective, it is inapplicable to our scenario.

c) LADRA. LADRA [21] judges the failure location and time according to the features and the failure cause according to the weight. It supports only four types of failures, namely,,,,,which are inconsistent with our scenario.

d) LogM, Ikeuchi et al. , Nagaraj et al. , Log3C, and Wang et al. LogM [22] proposes a framework that combines a correlation analysis approach and an anomaly knowledge graph, effectively aiding failure diagnosis in real scenarios. Ikeuchi et al. [24] utilize user actions to identify the root cause. Nagara et al. [23] propose to diagnose performance failures by comparing the logs of system behaviors in good and bad performance. Log3C [25] proposes a cascading clustering algorithm to group numerous log sequences promptly. Then, it identifies impactful problems that lead to Key performance indicator (KPI) degradation. Wang et al. [26] propose to correlating log anomaly information and KPI anomaly information. The log anomaly detection algorithm is used to obtain the log anomaly score sequence, which can be used to find failure-related metrics. These five works rely on the prior knowledge graph, user action, and KPIs, respectively, which cannot be obtained in our scenario.

e) LogCluster and Yuan et al. LogCluster [18] calculate log sequence vectors to represent log sequences and determines failure types by clustering. Yuan et al. [3] try to extracts representation vectors from anomalous logs and use these representation vectors to build a classifier to determine the failure type. However, the clustering method provided by LogCluster does not use the semantic information of the log itself. Although it does not need manual labels, the accuracy needs to be improved in practical applications. Yuan et al. [3] adopt supervised learning, which requires many manual labels to train a classifier.

B. NLP Concepts

Developers usually define logging statements in a natural language-like manner. LogKG utilizes some NLP tools to extract entities from the content field.

Open Information Extraction (OpenIE) [29] extracts semantic triples, which represents the information in the raw text, based on different NLP toolkits. LogKG uses OpenIE to extract semantic triples from log templates to represent the events contained in logs.

Stanford CoreNLP is an extensible pipeline that provides core natural language analysis [17]. It integrates various func-

TABLE II: Related work

Proposal	Objective	Data	Baselines	Reason
Yuan et al. [3] LogCluster [18]	Failure Diagnosis	Execution logs	Yes	The scenario of this work is the same as ours.
LogFlash [19] DeepLog [20]	Anomaly detection	Execution logs	No	Its output is the anomalous logs deviate from the position of the usual path.
LOGAN [9] Onion [7] FDiagV3 [8]	Failure Diagnosis	Execution logs	No	Its output is the log entries that potentially lead to the problem, which is inconsisteat with our objective.
LADRA [21]	Failure Diagnosis	Execution logs	No	This work supports only four types of failures, which is inconsisteat with our scenario.
LogM [22]	Failure Diagnosis	Execution logs Prior knowledge graph	No	This work heavily relies on prior knowledge about failure root causes and log events, which cannot be obtained in our scenario.
Nagaraj et al. [23]	Failure Diagnosis	Event logs State logs	No	This work relies on the classification of event logs and state logs, which cannot be obtained in our work.
Ikeuchi et al. [24]	Failure Diagnosis	Execution logs User actions	No	This work relies on user actions, which cannot be obtained in our scenario.
Log3C [25] Wang et al. [26]	Failure Diagnosis	Execution logs KPIs	No	This work relies on KPIs, which cannot be obtained in our scenario.

TABLE III: An example of parameter values

Template	Parameter Value
the list of all available services are VAR1: VAR2, VAR3: **VAR4**.	http://0.0.0.3:9388
	http://0.0.0.1:9386
	http://0.0.0.2:9387
now call service:**VAR1**, inst:VAR2 as a downstream service	redisservice1
	redisservice2

"VAR" corresponds to the parameter position in the log template. The second columu lists the parameter values of "VAR4" in the first template and "VAR1" in the second template.

TABLE IV: A template can represent multiple events

Template	Event
instance: VAR1 Starting instance _do_build_and_run_instance VAR2	instance is VAR1
	instance is Starting
VAR1 is a valid instance name _list_backing_images VAR2	VAR1 is instance name
	VAR1 is valid
	instance name is valid
image VAR1 at VAR2 in use	image is VAR1
	image VAR1 is at VAR2

tions such as NER (Named Entity Recognition), POS (part-of-speech), and OpenIE. LogKG performs triple extraction, keyword tagging, and parameter type identification based on CoreNLP to fully extract the information in the content field.

Common Event Expression (CEE) is a toolkit that offers a detailed taxonomy of general events. It improves the ability of users to effectively interpret and analyze events [30]. LogKG annotates the keywords in logs using CEE.

C. Challenges

Based on the above information, we introduce the three challenges lying in KG-based log failure diagnosis.

CH.1. Entity Extraction. As listed in Table IV, a log template can represent multiple events. For example, Table IV lists that "instance is VAR1" and "VAR1 is instance name" in the first template represent different events. Existing log-based event failure diagnosis methods treat each log template as a single entity, ignoring the multiple contextually relevant events contained in the log. So different events in a log need to be extracted as different event entities.

CH.2. Entity Alignment. Entity alignment aims for merging entities from different sources but semantically represent the same real-world object [31]. First, we need to identify the types of parameters to match the entities with the same type of parameters. However, there are many types of parameters, making rule-based parameter type identification impractical. For example, Table III lists the parameter values of two templates. Regular expressions can identify the types of the parameters in the two templates, i.e., Web URL and redis senice instance, respectively. However, as more services are deployed, there are too many parameters to set rules exhaustively. Second, we need to semantically align triple entities. For instance, in Table IV, "instance is VAR1" in the first template and "VAR1 is instance name" in the second template represent similar semantic events. Existing methods do not align those with similar semantics and merge them into the same entity.

CH.3. Failure Case Representation. In NLP tasks, representation vectors are usually calculated by statistical-based methods such as TF-IDF [32]. However, unlike natural language, logs usually contain much noise. For example, logs generated by scheduled tasks and normal business activities are usually irrelevant to failures and may interfere with failure diagnosis. These NLP methods are not suitable for obtaining failure-oriented log representations due to these noises

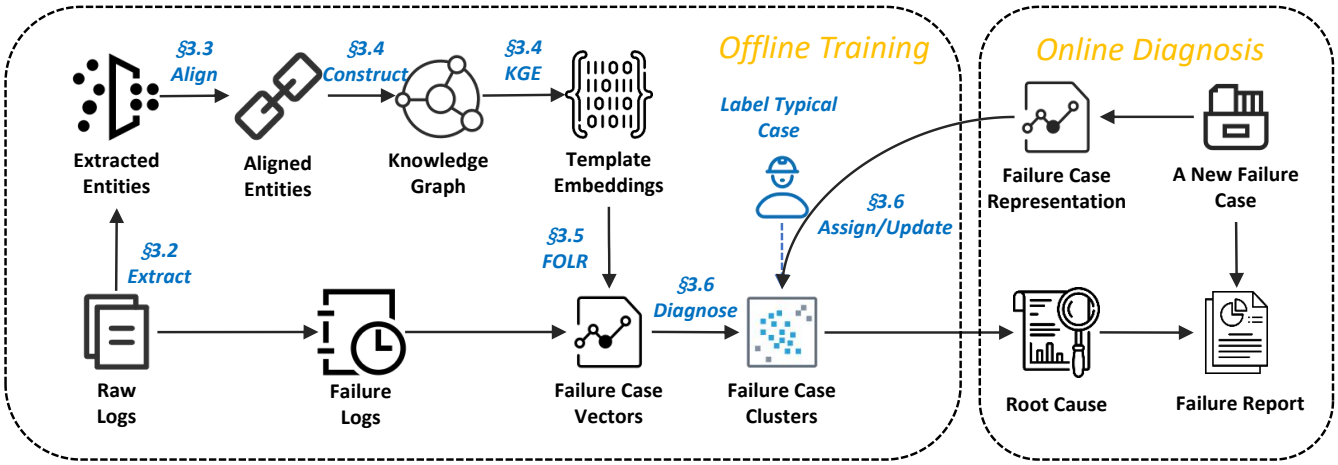


Fig. 2: The framework of LogKG

III. LogKG

A. The framework of LogKG

LogKG has two parts: offline training and online diagnosis, as shown in Figure 2. When extracting entities, it not only uses the information in the structured data but also extracts keywords, log triples, and parameters from the unstructured texts. It then aligns entities from different fields and fuses the multi-field information through a knowledge graph. It uses KGE to generate the representation vectors of failure cases. Then, it aggregates these failure cases by clustering. Operators label typical cases of these clusters. After a failure occurs, LogKG converts the failure-related logs into a representation vector and diagnoses the failure type based on the trained model.

Figure 3 shows the knowledge graph construction process, where the main steps are entity extraction and entity alignment. LogKG extracts triple from log templates in entity extraction, addressing the first challenge (*i.e.*, entity extraction). In entity alignment, LogKG aligns log triples based on semantics and identifies parameter types based on context, addressing the second challenge (*i.e.*, entity alignment).

Based on the above two stages, LogKG can extract and align entities from logs' multi-fields. Then it builds a knowledge graph to fuse the aligned entities. It then gets the representations of logs through KGE. Our proposed FOLR combines the local features of a failure case and the global features of all failure cases to find the failure-relevant logs. Then, it obtains the failure-oriented representation vector for each log template, solving the third challenge (*i.e.*, failure-oriented log representation).

B. Entity Extraction

As shown in Figure 3, a log can be divided into three parts: structured data, templates, and parameters. Based on these three parts, we can extract four entities, i.e., structured entities, parameters, keywords, and log triples. Structured entities can be extracted directly from logs. Parameters can be easily obtained through log parsing methods [10]. LogKG extracts keywords and log triples as follows.

1) Keyword Extraction: Log templates usually contain keywords related to components, states, and tasks in the system, such as "MySQL", "DataBase", "AMQP", "Error", etc. These keywords are vitally important for failure diagnosis. Therefore, they need to be extracted as entities. Specifically, LogKG utilizes CoreNLP [17] to select words from log templates based on part-of-speech tagging and use them as keywords.

2) Triple Extraction: As aforementioned, a log can contain multiple events. To accurately and efficiently extract the multiple events from logs, LogKG performs information extraction on log templates, extracting log triples contained in them. Specifically, it combines Rule Extraction (RE) and OpenIE [33] to extract the triples.

RE for Rule Triples. The purpose of RE is to take advantage of the regular structure of logs. According to our observation, there are some rules for systems to print logs. Therefore, it becomes easier to define rules to extract events precisely. For example, in our implementation, we use some rules to extract entity-value pairs because they are usually separated by a "=" or ":" symbol.

OpenIE for Semantic Triples. OpenIE [29] is often used to extract semantic triples. LogKG utilizes it to extract semantic triples from logs, representing the multiple events in the logs. There has been substantial progress in OpenIE approaches since it was proposed by Banko et al. [29]. These methods take free text as input and yield semantic triples as output, formed by two arguments related by a predicate such as ("Instance", "is", "valid"). Here, the implementation of OpenIE is not fixed.

Based on the above two components, LogKG extracts rule triples and semantic triples from log templates, respectively. In this way, each event is denoted as an entity through a triple.

C. Entity Alignment

Entity alignment aims to find entities from different sources but semantically representing the same real-world object. We need to align these entities and merge the aligned entities. In this way, we can build associations for the entities with underlying relationships through aligned entities. LogKG aligns keywords, log triples, and parameters as follows.

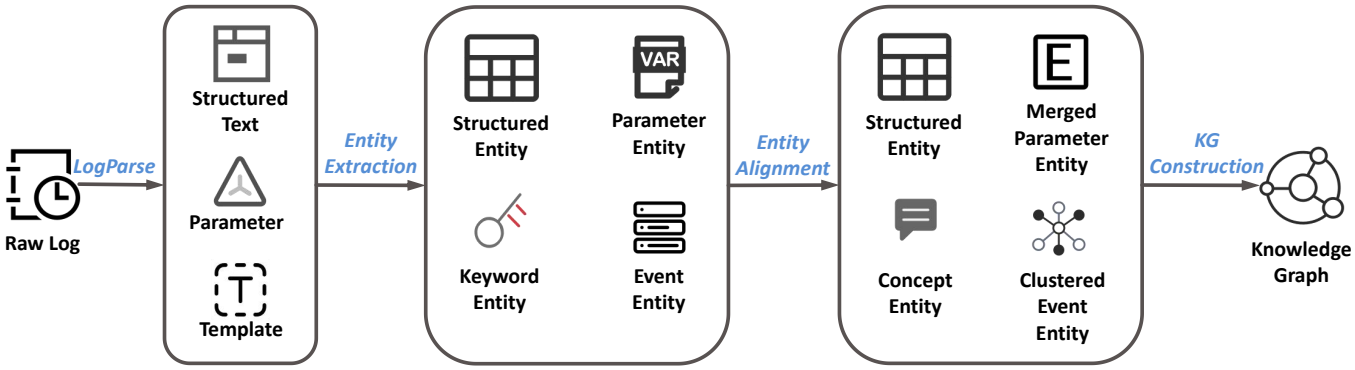


Fig. 3: The knowledge graph construction framework of LogKG

1) Keyword Alignment: For keyword entities, LogKG aligns them, integrating NLP tools and domain knowledge. Specifically, it semantically annotates keyword entities based on CEE library [30] and maps meaningful keywords into concept entities. However, the concepts contained in CEE are not comprehensive, so we extend the concepts of the CEE library based on domain knowledge. For example, the keyword "AMQP" can be annotated as "message middleware".

2) Triple Alignment: Although LogKG can automatically extract semantic triples from logs, the syntax of various triples is very different. Some triples representing similar semantics are pretty dissimilar. As listed in Table IV, the events "instance is VAR1" in the first template and "VAR1 is instance name" in the second template represent similar semantics. Logs corresponding to these two templates are context-related. In order to associate these context-related log templates, we need to build associations for these templates based on event entities. For example, two different templates can associate with the same event entity. More specifically, we need to align these triples based on semantics and aggregate semantically similar triples into one entity.

LogKG aggregate triples into clusters and treats each cluster as an event entity. To measure semantic similarity, it converts these triples into vectors first. Specifically, since each triple is a complete short sentence, LogKG utilizes a pre-trained language model to convert each triple into a fixed-dimensional vector. Some pre-trained language models based on BERT [34] have achieved superior performance in recent years. We choose a pre-trained language model [35] of them to obtain the vectors of these triples. Please note that the selection of pre-trained natural language models is not our main contribution. Then, LogKG aggregates these semantically similar triples into event entities by clustering. Since the number of clusters is pending, we choose some algorithms that do not require specifying the number of clusters. Here, the clustering algorithm can be DBSCAN [36], hierarchical clustering [37], OPTICS [38], or others.

3) Parameter Alignment: Unlike triples and keywords, parameters usually contain variable information of a specified type (e.g., IP address, filename, status code). It is necessary to identify the types of these parameters first. Then we can align these parameters representing the same type of object. To identify the type of parameters, LogKG utilizes its context information. First, LogKG uses regular expressions to identify the parameter type from its value. As listed in Table III, "http://0.0.0.2:9387" could be identified as the "Web URL" through regular expression. In addition, after entity extraction, we can extract the triples related to parameters, thereby identifying the parameter type with contextual information. As listed in the second template of Table III, the parameter "VAR1" corresponds to the "service" above. The RE component in LogKG can extract rule triple ("service", "is", "VAR1"), which means that the type of parameter "VAR1" is "service".

After identifying the types of parameters, the entities of the same type and representing the same object can be merged into one entity. For example, the structured entity "0.0.0.2" in the IP field can be merged with the parameter "http://0.0.0.2" whose type is "IP address".

D. Knowledge Graph Construction and Embedding

After entity extraction and entity alignment based on the multi-field information contained in the logs, LogKG constructs a knowledge graph based on the explicit relationships of different types of entities in logs. We focus on the log templates and establish direct or indirect relationships between the aligned entities and the log templates. For example, a log is associated with structured entities and parameter entities. Additionally, it is mapped to a template, which is associated with event entities and keyword entities. In this way, different entities from different fields in logs are associated with each other based on relationships.

Then, we need to transform the knowledge graph into a vector that can be applied to failure diagnosis. Knowledge graph embedding (KGE) embeds the components of a knowledge graph, including entities and relations, into continuous vector spaces [39]. Here, we adopt RotatE [40] as the KGE model. RotatE is a model with reasonable relationship inference performance. It can describe the potential relationship between various entities in the logs. Please note that choosing RotatE as the KGE model is not our contribution. We can replace it with other KGE models. We use the representation vectors of templates for failure diagnosis because, on the one hand, each log is mapped to a specific template. On the other hand, a log can be associated with two or more structured entities, parameter entities, keyword entities, or event entities.

E. Failure-Oriented Log Representations

For now, we have obtained the representation vector of each template. Next, we need to obtain the representation vector of each failure for failure diagnosis. Usually, some failure-irrelevant logs can appear multiple times during a failure. So we need to filter them out and use the remaining for failure representation. LogKG first obtains the logs during failure. The duration of failure that LogKG obtains is w, which varies with different types of services. According to our experience, we set w for 20 minutes. Then, LogKG can map these logs in the log set to log templates to constitute a template set.

TF-IDF [41] is a term weighting technique in information retrieval [18]. Motivated by TF-IDF, we proposed Failure-Oriented Log Representation (FOLR). According to our observation, logs that always appear in failure cases are usually not relevant to the failure. For each template, LogKG calculates its ILSF (Inverse Log Set Frequency) values. ILSF represents the frequency of each template in all failure cases. Templates that appear more frequently in failures usually have lower ILSF values. They are usually not relevant to failures. In order to obtain the failure-oriented templates, LogKG ignores log templates with lower ILSF values. Moreover, the same log template's occurrences differ in different types of failures. Therefore, for each template of a specific log set, LogKG calculates its TF (Template Frequency). Then, LogKG uses TF-ILSF to perform a weighted sum of the representation vectors of the log templates.

Formally, the ILSF of template $t(t \in T)$ is calculated as:

$$w_{ilsf}(t) = \begin{cases} log(\frac{N}{n_t}) & log(\frac{N}{n_t}) \geq \theta_{ilsf} \\ 0 & log(\frac{N}{n_t}) < \theta_{ilsf} \end{cases} \tag{1}$$

where N is the total number of failure cases, n_t denotes the number of failure cases where the template t appears, and θ_{ilsf} denotes the threshold of ILSF. We calculate the frequency of the occurrence of each template corresponding to a specific failure case as the TF value. TF of template $t(t \in T)$ of the i-th log set L_i is calculated as:

$$w_{tf}(t, L_i) = \frac{n_{t,i}}{n_i} \tag{2}$$

where $n_{t,i}$ is the number of times template t appears in log set L_i, n_i denotes the number of logs in log set L_i. By calculating TF and ILSF, LogKG can separately obtain the representation vector V_i of the log set L_i of each failure case using weighted summation:

$$V_i = \sum_{j=1}^{m} w_{tf}(T_j, L_i) \times w_{idf}(T_j) \cdot E_j \tag{3}$$

where T_j is the j-th template in T, m is the number of templates and E_j denotes the KGE of template T_j. After the above steps, LogKG can calculate each failure case's representation vector for diagnosis.

F. Diagnosis

Failure diagnosis is usually formalized as a classification task. It identifies the root causes of a failure [4]. LogKG trains a failure diagnosis model through clustering offline. In the online diagnosis phase, it assigns a failure to an existing cluster or updates the model.

1) Offline Training: To use the multi-field information fused by the knowledge graph in downstream tasks, LogKG first gets the representation vectors of log templates based on KGE, which are then used to obtain the representation vectors of failure cases. Subsequently, LogKG groups the representation vectors of failure cases into different clusters, and operators label the root cause of the typical cases for each cluster.

Failure clustering. We employ a density-based clustering method to cluster the representation vectors of failure cases. Both DBSCAN [42] and OPTICS [43] are widely used density-based clustering algorithms. After analyzing the representation vectors of the failure cases, the densities of different failure types are pretty different. Therefore, we adopted the OPTICS clustering algorithm since OPTICS is appropriate for our scenario, and it is insensitive to hyperparameters.

Failure diagnosis model. After clustering, operators select each cluster's most representative failure case. In detail, they usually select the one closest to its cluster centroid as the most representative failure. Afterward, they label the root causes of these cases as the root causes of their corresponding cluster. In this way, the labeling effort is much reduced because the number of failure clusters (ten categories in our scenario) is much smaller than that of failure cases.

2) Online Diagnosis: LogKG is triggered when a service failure is detected. It then collects the logs around the failure and calculates the failure representation vector as in the offline training stage. LogKG then assigns this failure to a failure cluster by calculating the distance between the calculated vector and the failure representation vectors of different failure clusters. Some new failure types often appear in the deployment process, so the failure diagnosis model needs to be updated. For some failures quite different from the existing ones, LogKG will update the clusters to adapt to the new failure types.

IV. Evaluation

A. Experiment Design

1) Datasets: We conduct experiments over the GAIA[2] dataset, a publicly available dataset, and the real-world log dataset collected from the production environment of CMCC, a top-tier global ISP. The detailed information of the two datasets is listed in Table V.

TABLE V: Detail of the datasets

Datasets	# of logs	# of failure categories	# of failure cases
GAIA	13,554,024	4	1,083
CMCC	1,461,006	6	93

a) **GAIA**: The GAIA dataset is generated by simulating real-word microservice failures. It contains the records of all failure injections, including the timestamp, location and all service logs of each failure. Specifically,

[2]https://github.com/CloudWise-OpenSource/GAIA-DataSet

13,554,024 logs are related to the 1083 failures, which can be classified into four types, including "login failure", "file uploading failure", "file moving failure" and "access permission denied exception".

b) **CMCC**: The CMCC dataset is collected from an OpenStack-based system of CMCC. OpenStack is the open-source cloud computing platform most widely adopted in the industry [44]. It consists of multiple service components, including four core components: Keystone for authentication, Nova for computing, Glance for image, and Neutron for networking. In addition, it also includes some supporting services, such as Database and Advanced Message Queue Protocol [44], each comprising multiple running service daemons. Based on the OpenStack framework, CMCC builds a 4G/5G core network to provide services for hundreds of millions of users with. We collect the 1,461,006 logs related to all the 93 failure cases in 24 days, which can be classified into 6 categories, including "AMQP server unreachable", "Mysql lost connection", "Computing nodes down", "Flavor disk too small", "Linuxbridge-agent anomalies" and "Nova-conductor lost connection". The failures are manually labelled by experienced operators.

In the following experiments, from either dataset, we leverage the front 70% failure cases as the training data and the rest 30% as the testing data.

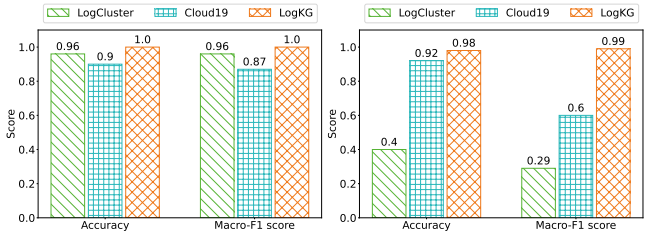
2) Baselines: We compare LogKG with two failure diagnosis algorithms: unsupervised algorithm LogCluster [18] and supervised algorithm Cloud19 [44]. The parameters of these methods are all set best for accuracy. Specifically, we choose

3) Experimental Setup: We conduct all the experiments on a PowerEdge-R740 server with 64 Intel Xeon Gold 5218 CPUs and 188GB of memory. We implement LogKG with Python 3.7 and PyTorch 1.8, and we implement LogCluster and Cloud19 with Python 3.7 and gensim (a free python library for NLP).

4) Evaluation Metrics: In our case, failure diagnosis can be viewed as a multiclassification problem. To measure the effectiveness of LogKG, we use the accuracy and the Macro-F1 score. Accuracy is the percentage of failure cases classified into the same category as the ground truth to all failure cases. To get a Macro-F1 score, we should get the F1 score for each failure category. For a particular failure category A, the precision is the percentage of failure cases with both classification result and ground truth being category A to all failure cases classified as A. The recall is the percentage of failure cases with both classification result and ground truth being category A to all failure cases with ground truth being category A. The F1 score of failure category A is the harmonic mean of precision(A) and recall(A). After we get the F1 score for all failure categories, the Macro-F1 score can be calculated by taking the average of the F1 score of all failure categories.

B. Evaluation of The Overall Performance

In this section, we compare LogKG with two baseline methods on two datasets to evaluate the effectiveness of our method. We choose OPTICS as the clustering algorithm.



(a) Score on CMCC (b) Score on GAIA

Fig. 4: The effectiveness of different methods on the two datasets

Figure 4 shows the comparison results of LogKG and two baseline methods on the CMCC and the GAIA datasets, respectively. Overall, LogKG achieves the best accuracy and Macro-F1 score among the three methods on both two datasets. More specifically, it has an accuracy of 1.0 and a Macro-F1 score of 1.0 on the CMCC dataset. Meanwhile, LogCluster and Cloud19 achieve accuracies of 0.96 and 0.90 and Macro-F1 scores of 0.96 and 0.87, respectively, which are lower than our method. The CMCC dataset contains fewer failure cases, and logs have simple patterns, so it is easy for all three methods to classify the cases into correct categories. However, even with such a task, the other two methods still suffer from false positives and false negatives.

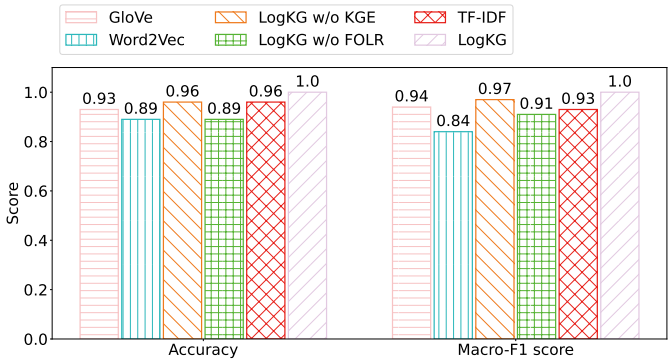
On the GAIA dataset, the effectiveness of the three methods is more clearly contrasted. LogKG has an accuracy of 0.98 and a Macro-F1 score of 0.99. When evaluating the performance of a multiclassification method, the shortcomings of the accuracy assessment method are particularly pronounced if the data is imbalanced. Although Cloud19 achieves an accuracy of 0.92, it has a low Macro-F1 score of 0.60, which shows that this method cannot classify every category of failure cases well. LogCluster achieves the lowest accuracy and Macro-F1 score on the GAIA dataset. Neither LogCluster nor Cloud19 thoroughly explores the multiple fields contained in the logs. At the same time, LogKG fuses the multi-field data of logs through knowledge graph. Therefore, our algorithm achieves superior failure diagnosis result than the two baseline methods.

C. Ablation Study

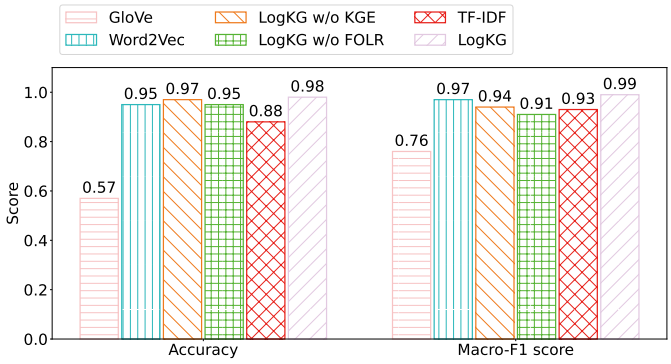
We conduct ablation experiments on two datasets to evaluate the effectiveness of two compulsory modules in LogKG: KG (knowledge graph) and FOLR (failure-Oriented log representations). Figure 5 shows the results of ablation experiments on the CMCC and GAIA datasets.

1) KGE: In our proposed method, we build a knowledge graph with multi-field data and use KGE to get the embedding of each log template, then we use the embedding of every log template of one failure case to get the whole embedding of this case. We believe the embedding obtained by this strategy integrates more log data information and is more effective. To evaluate the effectiveness of this strategy, we design the following ablation experiments:

a) We remove the KGE from LogKG and use the one-hot vector to embed each log template. More specifically,



(a) Score on CMCC



(b) Score on GAIA

Fig. 5: The effectiveness of LogKG when removing KGE, FOLR, or replacing them with different methods on the two datasets

suppose there are n log templates with indices from 1 to n in the whole dataset, then we use an n-dimensional vector to represent the i-th log template where the i-th bit in the vector is 1, and all other bits are 0. By doing so, we get the embedding of each log template, and the performance of this method is shown as "LogKG w/o KGE" in Figure 5.

b) We replace KGE with traditional algorithms in the NLP domain since getting an embedding of a sentence is a typical NLP task. We choose two typical NLP algorithms, GloVe and Word2Vec, to replace KGE and generate the embedding of each log template, respectively. The scores of LogKG with KGE replaced by GloVe and Word2Vec are shown as "GloVe" and "Word2Vec" in Figure 5, respectively.

From Figure 5, we can observe that KGE indeed improves the performance of LogKG. For example, LogKG without KGE achieves lower accuracy and lower Macro-F1 score on the two datasets than LogKG, respectively. When we replace KGE with Glove and Word2Vec, the results become less stable and, in most cases, not as good as just replacing LogKG with a one-hot vector. Log data is different from natural language text. It contains many repeated logs, and some logs do not conform to grammatical rules. Glove and Word2Vec can extract the semantic information of templates, but the above characteristics of the log data can make them inaccurate, degrading their performance. Although the one-hot method is simple, its statistical idea can be applied to log data, so it performs well. LogKG extracts entities from logs and builds a knowledge graph. It not only preserves the semantic information of logs but also mines the relationship between logs at a smaller granularity, which better adapts to the characteristics of logs. Thus we can conclude that LogKG can achieve better accuracy with KGE.

2) FOLR: We propose FOLR to generate the embedding of each failure from log template embedding. FOLR calculates each failure case's embedding by performing a weighted sum of the embeddings of log templates. Unlike TF-IDF, FOLR firstly performs denoising by discarding part of the logs with lower inverse document frequency which are considered noisy data. It uses a threshold to decide which logs to discard. In a real production environment, there are many noise logs during each failure, easily leading to inaccurate classification. To verify the effectiveness of FOLR, we design the following ablation experiments:

a) We remove FOLR from LogKG and use the sum of the embeddings of log templates to get the embedding of each failure case directly. LogKG w/o FOLR in Figure 5 shows the performance of this method.

b) We remove FOLR and use TF-IDF to generate the embedding of each failure case. "TF-IDF" in Figure 5 shows the performance of this method.

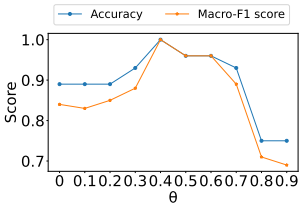
As shown in Figure 5, both LogKG without FOLR and LogKG with TF-IDF have lower accuracy and Macro-F1 scores on the two datasets than LogKG with FOLR. FOLR increases the Macro-F1 score by 0.08. The result indicates that FOLR is effective and improves the performance of LogKG.

D. Evaluation of Hyper-Parameters

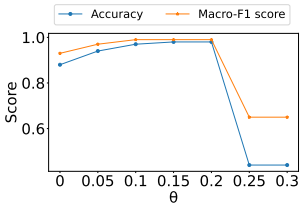
In this section, we evaluate the hyper-parameters of LogKG, which contains the threshold of FOLR (θ_{idf}), the $min\_samples$ (MS) of the clustering algorithm OPTICS, and the time window length (ω) of the failure case, as shown in Figure 6. A time window length ω indicates that we collect logs of $\omega/2$ minutes before and after the time when a failure occurs. We evaluate θ_{idf} and MS on two datasets and D on the CMCC dataset because we use the duration of each failure, which can be obtained in the dataset as, as the failure's corresponding time window.

CMCC dataset has fewer failure cases and simpler log patterns than the GAIA dataset, so parameter intervals are different in the figures. Specifically, we increase θ_{idf} from 0.00 to 0.30 with a step size of 0.05 on the CMCC dataset and from 0.00 to 0.90 with a step size of 0.1 on the GAIA dataset. As shown in Figure Figure 6(a) and Figure Figure 6(b), we find that as θ_{idf} increases, the Macro-F1 score tends to increase first and then decrease. The appropriate interval is from 0.40 to 0.50 on the CMCC dataset and from 0.10 to 0.20 on the GAIA dataset. LogKG will be influenced by more noisy data when choosing a small θ_{idf} and discard more useful logs when using a big one.

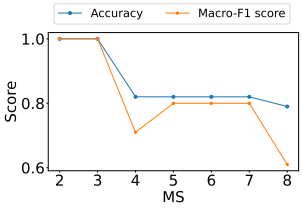
As for MS shown in Figure Figure 6(c) and Figure Figure 6(d), LogKG achieves better when $MS < 4$ on the CMCC dataset and when $MS < 12$ on the GAIA dataset. The results



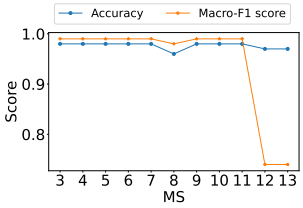
(a) The threshold of FOLR on CMCC



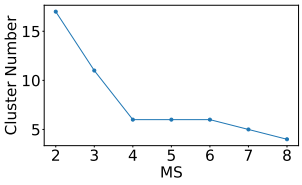
(b) The threshold of FOLR on GAIA



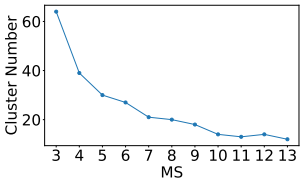
(c) The minimum number of samples of OPTICS on CMCC



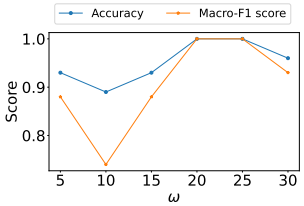
(d) The minimum number of samples of OPTICS on GAIA



(e) Cluster number as MS of OPTICS varies on CMCC



(f) Cluster number as MS of OPTICS varies on GAIA

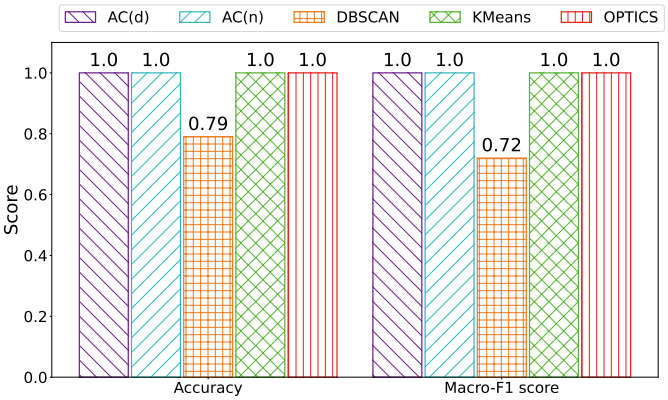


(g) Time window length of failure case on CMCC

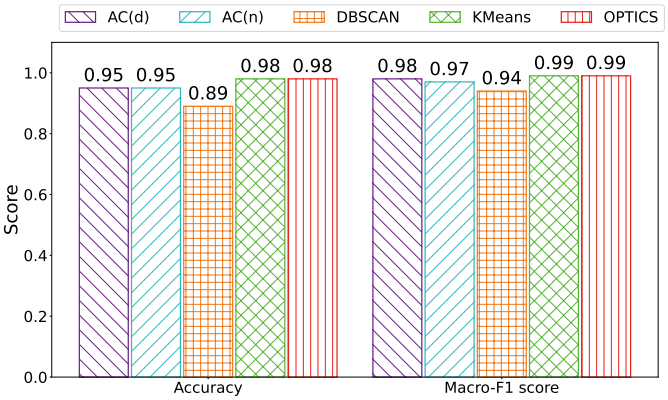
Fig. 6: Performance of LogKG as its parameters vary

are stable in most cases when MS takes a small value, and when the dataset is relatively small, selecting a larger MS will lead to an unstable clustering effect. Besides, we also count the number of clusters when MS varies since this indicator determines labor consumption. A large number of clusters bring more labeling work to operators. As shown in Figure Figure 6(e) and Figure Figure 6(f), the number of clusters is negatively correlated with the MS. It is less than 20 when MS changes on the CMCC dataset and has no more drastic changes when $MS > 7$ on the GAIA dataset. So we set their parameters when LogKG achieves higher accuracy and Macro-F1 score, i.e., $\theta_{idf} = 0.4$, $MS = 3$ on the CMCC dataset and $\theta_{idf} = 0.15$, $MS = 10$ on the GAIA dataset.

Figure Figure 6(g) shows how the time window length affects the performance of LogKG on the CMCC dataset. We increase ω from 5 to 30 with a step size of 5. We can find



(a) Score on CMCC



(b) Score on GAIA

Fig. 7: Performance of LogKG as its clustering algorithm change

that LogKG achieves the best performance when $\omega \in [20, 25]$. when ω is too small, some logs containing failure information will be missed, so the accuracy and Macro-F1 score may be poor and unstable. However, when ω is too big, more logs unrelated to the failure will be introduced, making LogKG less effective.

We also discuss the effect of different clustering algorithms on the two datasets. We conduct experiments with five different clustering algorithms: OPTICS, Agglomerative Clustering with specified distance threshold (AC(d)), Agglomerative Clustering with specified number of clusters (AC(n)), DBSCAN and KMeans, and the result is shown in Figure Figure 7(a) and Figure Figure 7(b), respectively. Except for DBSCAN, other clustering algorithms all achieve 1.0 on both accuracy and Macro-F1 score on the CMCC dataset and achieve more than 0.95 on both accuracy and Macro-F1 score on the GAIA dataset. DBSCAN achieves low accuracy and Macro-F1 score on the CMCC dataset and does not perform as well as the other four algorithms on the GAIA dataset. According to the experiments, OPTICS and KMeans have the best performance among the five algorithms, and we select OPTICS as the clustering algorithm in LogKG.

E. Threats to Validity

1) Data Quality: LogKG consists of two parts: the knowl-

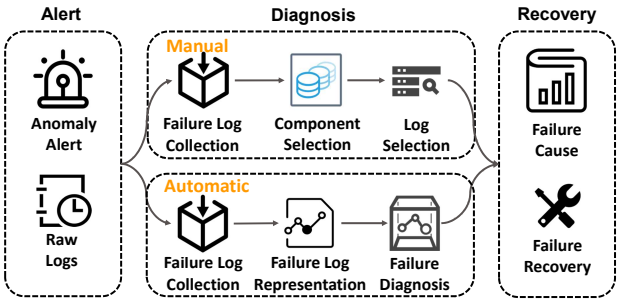


Fig. 8: Comparison of manual and automatic failure diagnosis.

edge graph construction and the failure diagnosis model training. The performance of the failure diagnosis model depends on the effect of the knowledge graph construction. In a real large-scale service scenario, some templates with poor quality may lead to incomplete extraction of multi-field information, affecting the effectiveness of knowledge graph construction. Therefore, in the face of complex templates or log formats, it is necessary to manually improve the quality of log templates. We believe these additional workloads do not require excessive human efforts.

2) Scope of Failure Logs: In our experiments, we set different failure log duration for the GAIA dataset and the CMCC dataset, respectively. For the GAIA dataset, since the duration of the failure is provided, we collect the logs of each failure case according to the information. For the CMCC dataset, since only the approximate time of each failure case is given, we use the time within five minutes before and after the given time as the duration of the failure. However, in real scenarios, the duration of a failure case is usually not fixed, and the recorded alarm time may also have a deviation. The above situations may affect the range of the failure logs, thereby affecting failure diagnosis performance. Since the amount of data in our experiments is still limited, it is impossible to verify the impact of the above situations. We will verify it on more online large-scale services in future work.

V. Deployment & Case Study

We have deployed LogKG in CMCC to verify its performance in knowledge graph construction and failure diagnosis. As shown in Figure 8, operators first collect failure logs and determine the influence scope of the failure in the manual failure diagnosis process. Then, they find failure-related logs out and diagnose the root cause of the failure based on domain knowledge. They usually need to specify rules to view the log context and have to search a large number of logs manually. On the one hand, it is inefficient to search logs through keywords manually. On the other hand, it is error-prone to diagnose the root cause of failures based on experience or rules.

Based on LogKG, operators do not need to do a lot of error-prone and inefficient log searching. LogKG shortens failure mitigation time and increases failure diagnosis accuracy. In this section, we first introduce the deployment of LogKG in CMCC. Then we illustrate the knowledge graph construction and failure diagnosis process through real-word examples.

A. Deployment in CMCC

1) Environment of CMCC: CMCC is a top-tier global ISP which provides customers with efficient and reliable communication service support. Based on NFV [45] technology, CMCC has built a 4G/5G core network, which provides calling and Internet services for hundreds of millions of users. OpenStack [44] is the core facility of NFV technology.

2) Workflow: Figure 9 shows the workflow in the production environment of CMCC including three steps: The log-based anomaly detection and failure diagnosis work in the production environment are divided into three steps.

Step. 1: Collection and Processing. CMCC uses Filebeat [46] to aggregate log files from different service instances and writes log streams to Kafka [47]. Operators connect Kafka with Logstash [48], which preprocesses logs utilizing extensions and plugins Then, operators match the logs to templates using log parsing method. The logs can be processed and stored in the database.

Step. 2: Failure Diagnosis. When a failure, LogKG automatically collects logs . Subsequently, LogKG computes the representation vector of the failure based on these logs and uses the clustering model trained offline to diagnose the cause of the failure.

Step. 3: Report and Notification After LogKG diagnosing the failure, the system will generate a failure report based on the failure-related logs and the diagnosed failure cause. It includes the alarm results of anomaly detection, the possible root causes of failures diagnosed by LogKG, and suggestions for handling based on historical experience. Then, it will be displayed on the web page and sent to the operators via SMS or email.

3) Performance: Specifically, we have deployed LogKG in Guangdong Mobile Communications Co., Ltd (GMCC). LogKG diagnoses the failure of more than 2000 nodes, which generate hundreds of millions of logs daily. During the five months of deploying it for failure diagnosis, LogKG averagely processed 47 cloud-based network failures daily. Based on LogKG, the failure delimitation and location can be completed within five minutes, and the average failure repairing time has been reduced by more than 20 minutes.

B. Knowledge Graph Construction

1) Graph Databases Stored in Application Scenarios: The application scenario in this paper requires the storage of large-scale unstructured data and the display of multi-field data association relations, so Neo4j which is based on graph structure storage is applied. It has excellent performance of large-scale data Query and flexible storage of unstructured data. It includes a Neo4j data browser to execute CQL (Cassandra Query Language) instruction and supports the visual interface.

2) The Three-Stage Construction Process:

Generally speaking, the construction of a knowledge graph usually goes through three stages: entity extraction, entity pairing, and graph construction. Figure 10 shows the final result of the log information in the graph database.

Step. 1: Entity Extraction. The entities, attributes, and relationships between entities are extracted from various types

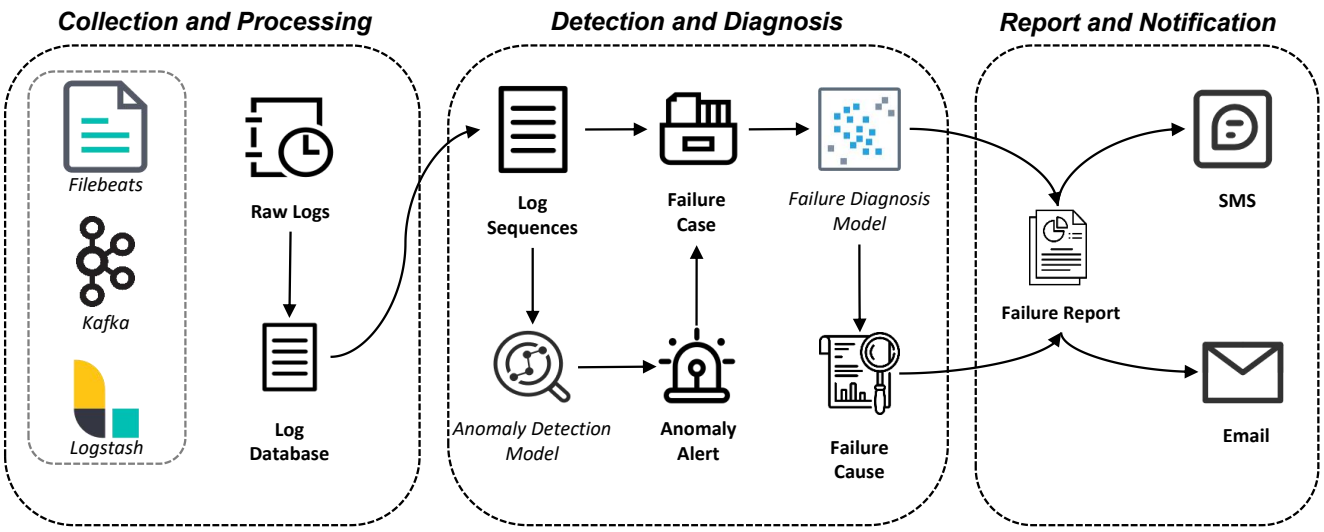


Fig. 9: The workflow in industrial scenarios of CMCC.

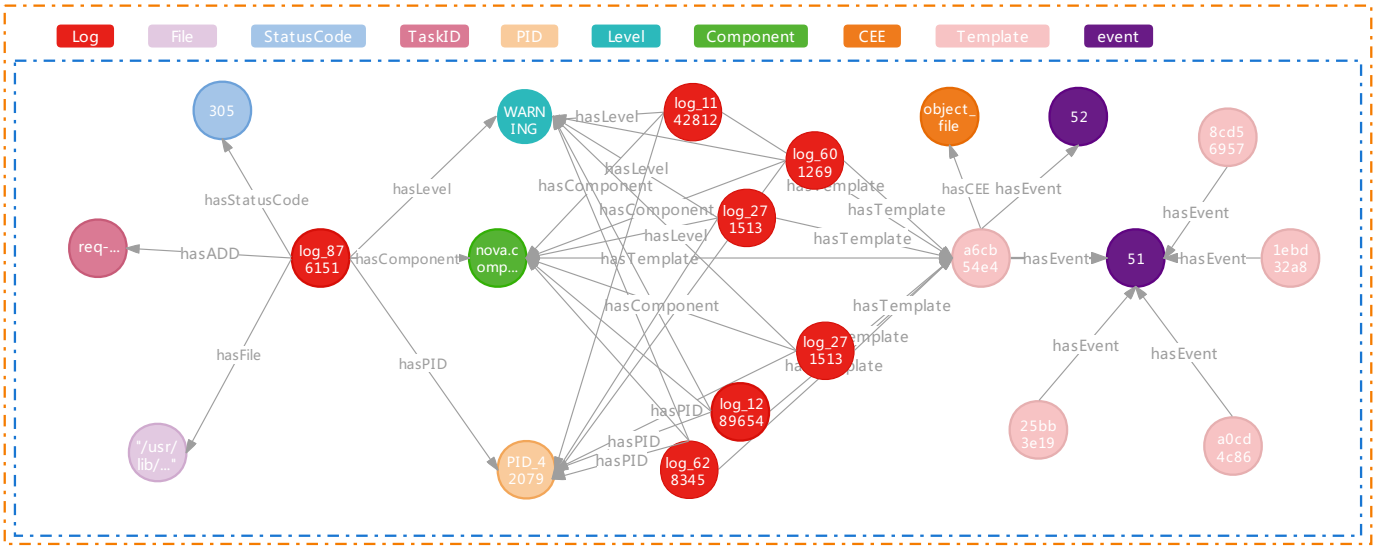


Fig. 10: Snippet of constructed log knowledge graph(best viewed in color).

of log data sources, and triplet-based knowledge representation is formed. The specific process is to extract the parsed structured text, parameters, and templates through logs to obtain structured entities, log triple, parameter entities, and keyword entities. The structured entities include “PID”, “ADDR”, and the parameter entities mainly consist of “File”, “StatusCode”, “IP”, the log triple mainly include “Event”.

Step. 2: Entity Alignment. The alignment operation is after entity extraction. structured entities are retained as structured entities. In order to build as many indirect associations between different entities as possible, we need to align these entities and fuse the aligned entities. For example, the keywords mention the information MySQL, SqlServer, and Access, they all correspond to the database concept and are unified and aggregated into one entity. Different types of entities align in different ways. some entities with potential relationships can build associations based on aligned entities.

Step. 3: Graph Construction. After entity extraction and entity alignment based on the multi-field information contained in the logs, the knowledge graph will be constructed based on the explicit relationships of several types of entities in the log. The triplet data we stored in Neo4j is parsed from 3223438 logs, which are 24-day logs from CMCC. Figure 10 shows a randomly selected subgraph, with the different node-types, each with label as the corresponding log textual description extracted. We can see in the figure that LogKG will automatically associate multi-field information of different logs such as “TaskID”, “Template”, “Param” (including file, status code, IP, etc) , “Level”, “PID”, “Component”, “CEE”. We use the relationships to establish associations between entities. We construct the logs into the above entity information and its corresponding relationships.

In conclusion, the construction of the knowledge graph makes full use of multi-field data to model the correlation of different types of semantic information in logs.

C. Case Study

Here, we use two selected failure cases to compare the amount of work required using LogKG and manual keyword search respectively. They are with different root causes, one was caused by the shutdown of the MySQL component, and the other was due to the exception of the Nova Conductor component. As listed in Table.VI, we count the number of logs that need to be investigated during the above two failure cases with different keywords. As mentioned above, support services such as MySQL and AMQP are used in OpenStack to provide functions such as database and communication. When the MySQL service is shut down, some services may fail due to the unavailability of the MySQL service during execution. For example, "Unexpected error while reporting service status" will be printed in the business log. Although the operators can use the keyword search method to find out the failure information provided by these logs, it requires a lot of manual work. For instance, they may have to examine 2500+ logs to find the log indicating the root cause.

However, the manual effort required is significantly reduced by LogKG. After detecting a failure in the online production environment, LogKG can calculate the representation vector of this failure automatically. Then the failure diagnosis model trained offline is used to classify it into the known root causes and generate a failure report.the whole process is completed in less than without any manual effort.

TABLE VI: The number of logs containing different keywords during two failure cases

failure Case	Kill	Fail	Error	Exception
Nova Conductor Error	157	48	3	4
MySQL Shutdown	322	15	811	1449

VI. Conclusion

In real scenarios, when large-scale services face more complex failure diagnosis, network equipment manufacturers need to use the complex rules hidden in logs to extract complete and standardized multi-field information to realize automatic failure diagnosis instead of excessive manual analysis. This improves the security and reliability of the service. In this paper, we use multi-field information fusion modeling in logs to automatically construct a knowledge graph, which can extract more information from the data for classification decisions. We use the knowledge graph information to train the failure diagnosis model, and finally know the cause of the failure. In evaluation experiments using real-world datasets, we found that the random forest classifier and optical clustering algorithm significantly improve the accuracy and Macro-F1 score performance of the LogKG. We hope that better clustering methods will be discussed in the future.

References

[1] W. Meng, Y. Liu, Y. Zhu, S. Zhang, D. Pei, Y. Liu, Y. Chen, R. Zhang, S. Tao, P. Sun *et al.*, "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs." in *IJCAI*, vol. 19, no. 7, 2019, pp. 4739–4745.

[2] S. Huang, Y. Liu, C. Fung, R. He, Y. Zhao, H. Yang, and Z. Luan, "Hitanomaly: Hierarchical transformers for anomaly detection in system log," *IEEE Transactions on Network and Service Management*, vol. 17, no. 4, pp. 2064–2076, 2020.

[3] Y. Yuan, W. Shi, B. Liang, and B. Qin, "An approach to cloud execution failure diagnosis based on exception logs in openstack," in *2019 IEEE 12th International Conference on Cloud Computing (CLOUD)*, 2019, pp. 124–131.

[4] S. He, P. He, Z. Chen, T. Yang, Y. Su, and M. R. Lyu, "A survey on automated log analysis for reliability engineering," *ACM Computing Surveys (CSUR)*, vol. 54, no. 6, pp. 1–37, 2021.

[5] C. Deng, Y. Jia, H. Xu, C. Zhang, J. Tang, L. Fu, W. Zhang, H. Zhang, X. Wang, and C. Zhou, "Gakg: A multimodal geoscience academic knowledge graph," in *Proceedings of the 30th ACM International Conference on Information & Knowledge Management*, 2021, pp. 4445–4454.

[6] A. V. Kannan, D. Fradkin, I. Akrotirianakis, T. Kulahcioglu, A. Canedo, A. Roy, S.-Y. Yu, M. Arnav, and M. A. Al Faruque, "Multimodal knowledge graph for deep learning papers and code," in *Proceedings of the 29th ACM International Conference on Information & Knowledge Management*, 2020, pp. 3417–3420.

[7] X. Zhang, Y. Xu, S. Qin, S. He, B. Qiao, Z. Li, H. Zhang, X. Li, Y. Dang, Q. Lin *et al.*, "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*, 2021, pp. 1253–1263.

[8] E. Chuah, A. Jhumka, J. C. Browne, B. Barth, and S. Narasimhamurthy, "Insights into the diagnosis of system failures from cluster message logs," in *2015 11th European Dependable Computing Conference (EDCC)*. IEEE, 2015, pp. 225–232.

[9] B. C. Tak, S. Tao, L. Yang, C. Zhu, and Y. Ruan, "Logan: Problem diagnosis in the cloud using log-based reference models," in *2016 IEEE International Conference on Cloud Engineering (IC2E)*. IEEE, 2016, pp. 62–67.

[10] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *2017 IEEE international conference on web services (ICWS)*. IEEE, 2017, pp. 33–40.

[11] Y. Liu, X. Zhang, S. He, H. Zhang, L. Li, Y. Kang, Y. Xu, M. Ma, Q. Lin, Y. Dang *et al.*, "Uniparser: A unified log parser for heterogeneous log data," *arXiv preprint arXiv:2202.06569*, 2022.

[12] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li *et al.*, "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*, 2019, pp. 807–817.

[13] Y. Zuo, Y. Wu, G. Min, C. Huang, and K. Pei, "An intelligent anomaly detection scheme for micro-services architectures with temporal and spatial data analysis," *IEEE Transactions on Cognitive Communications and Networking*, vol. 6, no. 2, pp. 548–561, 2020.

[14] F. Liu, Y. Wen, D. Zhang, X. Jiang, X. Xing, and D. Meng, "Log2vec: A heterogeneous graph embedding based approach for detecting cyber threats within enterprise," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1777–1794.

[15] F. Wang, A. Bundy, X. Li, R. Zhu, K. Nuamah, L. Xu, S. Mauceri, and J. Z. Pan, "Lekg: A system for constructing knowledge graphs from log extraction," in *The 10th International Joint Conference on Knowledge Graphs*, 2021, pp. 181–185.

[16] A. Ekelhart, F. J. Ekaputra, and E. Kiesling, "The slogert framework for automated log knowledge graph construction," in *European Semantic Web Conference*. Springer, 2021, pp. 631–646.

[17] C. D. Manning, M. Surdeanu, J. Bauer, J. R. Finkel, S. Bethard, and D. McClosky, "The stanford corenlp natural language processing toolkit," in *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, 2014, pp. 55–60.

[18] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *2016 IEEE/ACM 38th International Conference on Software Engineering Companion (ICSE-C)*. IEEE, 2016, pp. 102–111.

[19] T. Jia, Y. Wu, C. Hou, and Y. Li, "Logflash: Real-time streaming anomaly detection and diagnosis from system logs for large-scale software systems," in *2021 International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2021.

[20] M. Du, F. Li, G. Zheng, and V. Srikumar, "Deeplog: Anomaly detection and diagnosis from system logs through deep learning," in *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, 2017, pp. 1285–1298.

[21] S. Lu, B. Rao, X. Wei, B. Tak, L. Wang, and L. Wang, "Log-based abnormal task detection and root cause analysis for spark," in *2017 IEEE International Conference on Web Services (ICWS)*. IEEE, 2017, pp. 389–396.
[22] Y. Xie, K. Yang, and P. Luo, "Logm: Log analysis for multiple components of hadoop platform," *IEEE Access*, vol. 9, pp. 73 522–73 532, 2021.
[23] K. Nagaraj, C. Killian, and J. Neville, "Structured comparative analysis of systems logs to diagnose performance problems," in *9th USENIX Symposium on Networked Systems Design and Implementation (NSDI 12)*, 2012, pp. 353–366.
[24] H. Ikeuchi, A. Watanabe, T. Kawata, and R. Kawahara, "Root-cause diagnosis using logs generated by user actions," in *2018 IEEE Global Communications Conference (GLOBECOM)*. IEEE, 2018, pp. 1–7.
[25] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 60–70.
[26] L. Wang, N. Zhao, J. Chen, P. Li, W. Zhang, and K. Sui, "Root-cause metric location for microservice systems via log anomaly detection," in *2020 IEEE International Conference on Web Services (ICWS)*. IEEE, 2020, pp. 142–150.
[27] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, and N. Bidokhti, "How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 200–211.
[28] J. Lu, F. Li, L. Li, and X. Feng, "Cloudraid: hunting concurrency bugs in the cloud via log-mining," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2018, pp. 3–14.
[29] M. Banko, M. J. Cafarella, S. Soderland, M. A. Broadhead, and O. Etzioni, "Open information extraction from the web," in *CACM*, 2008.
[30] "Mitre: Common event expression." Website, 2014, https://cee.mitre.org/.
[31] K. Zeng, C. Li, L. Hou, J. Li, and L. Feng, "A comprehensive survey of entity alignment for knowledge graphs," *AI Open*, vol. 2, pp. 1–13, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2666651021000036
[32] J. Ramos *et al.*, "Using tf-idf to determine word relevance in document queries," in *Proceedings of the first instructional conference on machine learning*, vol. 242, no. 1. Citeseer, 2003, pp. 29–48.
[33] G. Stanovsky, I. Dagan *et al.*, "Open ie as an intermediate structure for semantic tasks," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2015, pp. 303–308.
[34] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.
[35] N. Reimers and I. Gurevych, "Making monolingual sentence embeddings multilingual using knowledge distillation," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2020. [Online]. Available: https://arxiv.org/abs/2004.09813
[36] K. Khan, S. U. Rehman, K. Aziz, S. Fong, and S. Sarasvady, "Dbscan: Past, present and future," in *The fifth international conference on the applications of digital information and web technologies (ICADIWT 2014)*. IEEE, 2014, pp. 232–238.
[37] F. Murtagh and P. Contreras, "Algorithms for hierarchical clustering: an overview," *Wiley Interdisciplinary Reviews: Data Mining and Knowledge Discovery*, vol. 2, no. 1, pp. 86–97, 2012.
[38] J. C. Dassun, A. Reyes, H. Yokoyama, M. Dolendo *et al.*, "Ordering points to identify the clustering structure algorithm in fingerprint-based age classification," *Virtutis Incunabula*, vol. 2, no. 1, pp. 17–27, 2015.
[39] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.
[40] Z. Sun, Z.-H. Deng, J.-Y. Nie, and J. Tang, "Rotate: Knowledge graph embedding by relational rotation in complex space," in *International Conference on Learning Representations*, 2019. [Online]. Available: https://openreview.net/forum?id=HkgEQnRqYQ
[41] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.
[42] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu, "Density-based spatial clustering of applications with noise," in *Int. Conf. Knowledge Discovery and Data Mining*, vol. 240, 1996, p. 6.
[43] M. Ankerst, M. M. Breunig, H.-P. Kriegel, and J. Sander, "Optics: Ordering points to identify the clustering structure," *ACM Sigmod record*, vol. 28, no. 2, pp. 49–60, 1999.
[44] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 366–367.
[45] J. d. J. G. Herrera and J. F. B. Vega, "Network functions virtualization: A survey," *IEEE Latin America Transactions*, vol. 14, no. 2, pp. 983–997, 2016.
[46] B. Elasticsearch, "Filebeat-lightweight shipper for logs (2020)," *URL https://www. elastic. co/products/beats/filebeat. Accessed*, pp. 02–12, 2021.
[47] J. Kreps, N. Narkhede, J. Rao *et al.*, "Kafka: A distributed messaging system for log processing," in *Proceedings of the NetDB*, vol. 11, 2011, pp. 1–7.
[48] E. Stack, "Elasticsearch, logstash, kibana— elastic," *URL: https://www.elastic.co/what-is/elk-stack*, 2021.

Yicheng Sui received the B.S. degree in Software Engineering from Nankai University in 2020. He is currently working toward the Ph.D. degree at the College of Software, Nankai University. His current research interests include machine learning and deep learning.

Yuzhe Zhang received the B.S. degree in Software Engineering from Nankai University in 2020. He is currently a M.S. student in the College of Software, NanKai University. His research interests include deep learning and anomaly detection.

Jianjun Sun received B.S. in semiconductor physics and devices from Tsinghua University, Beijing, China, in 1991 and M.S. in radio electronics from Jinan University, Guangzhou, China, in 1994. He is currently the general manager of the Network Management Center of China Mobile Communications Corporation Guangdong Co., LTD His current research interests include communication network maintenance management and NFV cloud network architecture.

Shenglin Zhang received B.S. in network engineering from the School of Computer Science and Technology, Xidian University, Xi'an, China, in 2012 and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2017. He is currently an associate professor with the College of Software, Nankai University, Tianjin, China. His current research interests include failure detection, diagnosis and prediction for service management. He is an IEEE Member.

Zhengdan Li received the M.E. degree in Software Engineering from Nankai University in 2020. She is an assistant experimentalist of College of Software, Nankai University. Her research interests include Artificial Intelligence, Software Engineering, etc.

Yongqian Sun received the B.S. degree in statistical specialty from Northwestern Polytechnical University, Xi'an, China, in 2012, and Ph.D. in computer science from Tsinghua University, Beijing, China, in 2018. He is currently an assistant professor with the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection and root cause localization in service management.

Fangrui Guo is currently a M.E. student in the College of Software, Nankai University. Her research interests include anomaly detection and failure diagnosis.

Junyu Shen received the B.S. degree in Software Engineering from Nankai University in 2022. is currently a M.S. student in the College of Software, Nankai University. His current research interest include anomaly detection and natural language processing.

Yuzhi Zhang received the B.S. and M.S. degree in computer science from the Department of Computer Science and Technology, Tsinghua University in 1985 and 1987, respectively, and the Ph.D. degree in computer science from the Institute of Computing Technology, Chinese Academy of Sciences in 1991. He is currently dean of the College of Software, Nankai University, and is also a distinguished professor. His research interests include deep learning and other aspects in artificial intelligence.

Dan Pei received the B.E. and M.S. degree in computer science from the Department of Computer Science and Technology, Tsinghua University in 1997 and 2000, respectively, and the Ph.D. degree in computer science from the Computer Science Department, University of California, Los Angeles (UCLA) in 2005. He is currently an associate professor in the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general. He is an IEEE senior member and an ACM senior member.