LogKG: 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,* Xiao Yang, 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 reliability and security. The current automatic log failure diagnosis methods cannot fully use the multiple fields of logs, which fail to capture the relation between them. In this paper, we propose LogKG, a new framework for diagnosing failures based on knowledge graphs (KG) of logs. LogKG fully extracts entities and relations from logs to mine multi-field information and their relations through the KG. To fully use the information represented by KG, we propose a failure-oriented log representation (FOLR) method to extract the failure-related patterns. 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, respectively, 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.

1 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], e.g., network link down, software bug, hardware crash, or misconfiguration. 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, which is vitally essential for the system's security and reliability. They are semi-structured text generated by logging statements in source code. A log has multiple fields, including timestamp, level, content, IP, component, task ID, and content, representing when it is generated, the severity of the event, the detailed event information, and the device, software component, and process that generate it, respectively. The unstructured content field usually consists of sentences predefined by developers in the source code to represent a specific event or state information during the program's execution. The other fields are structured. Table 1 lists several examples of logs in the Generic AIOps Atlas (GAIA) dataset [5]. According to the experience of operators, it is necessary to consider the information from multiple fields in manual failure diagnosis (see §2.2.3 for more details). Such as Figure 1 shows the logs during two failures. The logs highlighted in red indicate the root causes

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.

- *S. Zhang is the corresponding author.*
- *Y. Sui, Y. Zhang, T. Xu, Z. Li, Y. Sun, and J. Shen are with Nankai University, Tianjin, China. Email: {lzd, sunyongqian}@nankai.edu.cn, {suiyicheng, zyzcs, xuting, 2120220688 }@mail.nankai.edu.cn.*
- *J. Sun, X. Yang, and L. Yu are with China Mobile Communications Corporation. Email:sunjianjun@gd.chinamobile.com, {yangxiaoyjy, yuliyf}@chinamobile.com*
- *S. Zhang and Y. Zhang are with the College of Software, Nankai University, Tianjin, China, Key Laboratory of Data and Intelligent System Security, Ministry of Education, China, and also with the Haihe Laboratory of Information Technology Application Innovation (HL-IT), Tianjin, China. Email: {zhangsl, zyz}@nankai.edu.cn*
- *F. Guo is with Accumulus Technology (China) Co., Ltd. Email: fangrui.guo@yunzhanghu.com*
- *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.*
- *This work was supported in part by the Advanced Research Project of China (No. 31511010501), National Natural Science Foundation of China (Grant No. 62272249, 62072264), and Natural Science Foundation of Tianjin (Grant No. 21JCQNJC00180).*

TABLE 1: 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

TABLE 2: Entities And Relations In GAIA Dataset

Subject	Predicate	Object
Log	*hasTemplate*	Template
Log	*hasComponent*	Component
Log	*hasPID*	PID
Log	*hasLevel*	Level
Log	*hasRequestID*	RequestID
Template	*hasEvent*	Event
Template	*hasCEE*	CEE

of these failures, which we call failure-related logs. It is challenging to find them from the raw logs by simply using the content field. When information from other fields is considered holistically, things get easier. The failure-related logs for the first case in Figure 1 can be 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, integrating the task ID "d04ac67cf6c285cd", the components "redisservice1" and "dbservice1", and the content field makes it to discover the failure-related logs. They can be used to determine the failures' root causes.

Some log-based failure diagnosis methods [3], [6]–[10] mine the information contained in logs. However, they only focus on the unstructured content field, neglecting the information of the structured fields, making them unable to fully mine the failures' features. Existing studies [6], [7] have shown the knowledge graph (KG) technique can effectively fuse unstructured texts and structured data. Therefore, we fuse the unstructured content field and structured fields (e.g., timestamp, level, IP, component, and task ID) of logs using and utilizing a KG for failure diagnosis.

A KG uses entities and relations to mine information in multiple fields. Existing KG-based text fusion methods [7] usually consists of three steps: *entity extraction, entity alignment, and relation extraction*. The information in the structured fields, as listed in Table 1, is relatively fixed and straightforward. We can easily extract the entities from them. For the unstructured texts, the existing log parsing methods [11], [12] can effectively resolve the texts into templates and parameters. Log templates can be used to extract and align the entities. Then, entities with different representations referring to the same object can be aggregated into the same entity in the KG. As for relation extraction, there are only several relations between logs and entities extracted from structured fields, such as 'hasComponent", "hasPID", "hasLevel", and "hasRequestID". The relations between templates and the entities extracted from templates are "hasCEE" and "hasEvent", etc. Moreover, another primary relation is "hasTemplate" between a log and a template. Table 2 lists the types of entities and relations in the KG built based on the GAIA dataset and the triple prototypes of them. To sum up, the KG constructed based on logs only has several pre-defined types of relations that can be easily obtained. Thus, the primary task of multi-field information fusion for logs is to extract entities from various fields and align them based on semantics. In this way, the entities and relations included in logs can be retrieved, and we can construct a KG to fuse the multi-field information.

To overcome the challenges lying in entity extraction, entity alignment, and failure representation (more details in section §2.3), we propose LogKG, a novel framework for failure diagnosis based on KG-based multi-field information fusion. LogKG extracts different types of entities and relations from the multiple fields of logs and aligns the entities based on semantics. It represents logs utilizing KG embedding (KGE) and obtains failure-oriented log representation vectors (FOLR) for failure cases. Then it aggregates historical failure cases and labels the root causes of the typical failure cases for each cluster. Above all, the following are the primary contributions of this paper:

Log Event Extraction. To extract entities from the multiple fields of logs, we propose a log event extraction method through 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 KG construction.

Alignment for Triples. An entity alignment method can aggregate entities related to the same object and establish indirect associations between them from multiple fields of logs. A semantics-based clustering method generates BERT [13] variables to obtain the vectors of log triples and merges triples with the same semantics into one entity to align them.

Failure-Oriented Log Representations. To mine failure features, we propose FOLR to extract failure representation vectors from logs. It finds failure-related logs after a failure occurs and calculates the failure's representation vector.

Published Dataset. We publish the failure case dataset[1] that we use, which is collected from China Mobile Communications Corporation (CMCC) , a top-tier global Internet Service Provider (ISP). These logs come from different types of faults in real scenarios and support the subsequent research on log-based failure diagnosis. We also publish our source code in the same repository.

1. https://anonymous.4open.science/r/LogKG-A6BD

TABLE 3: Related Work

Proposal	Objective	Data	Baselines	Reason
Yuan et al. [3] LogCluster [14]	Failure Diagnosis	Execution logs	Yes	The scenario of these works is the same as ours.
LogFlash [15] DeepLog [16]	Anomaly detection	Execution logs	No	Their outputs are the anomalous logs that deviate from the position of the usual path, which is inconsistent with our objective.
LOGAN [10] Onion [8] FDiagV3 [9]	Failure Diagnosis	Execution logs	No	Their outputs are the log entries that potentially lead to the problem, which are inconsistent with our objective.
LADRA [17]	Failure Diagnosis	Execution logs	No	This work supports only four types of failures, which is inconsistent with our scenario.
LogM [18]	Failure Diagnosis	Execution logs Prior KG	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. [19]	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. [20]	Failure Diagnosis	Execution logs User actions	No	This work relies on user actions, which cannot be obtained in our scenario.
Log3C [21] Wang et al. [22]	Failure Diagnosis	Execution logs KPIs	No	These works rely on KPIs, which cannot be obtained in our scenario.

To measure the effectiveness of LogKG, we evaluate it on a real-world dataset collected from CMCC and a popular open-source GAIA dataset. LogKG improves the accuracy and Macro-F1 scores by 4% and 1.5% over the two baselines on the CMCC dataset, respectively. At the same time, the improvements on the GAIA dataset are 26% and 15%, respectively. We also demonstrate the effectiveness of FOLR and KG through ablation studies.

Based on a 5-month deployment of LogKG in CMCC, we evaluate the workloads of manual verification and automatic failure diagnosis based on LogKG, respectively. LogKG handles an average of 47 service failures per day, which can be diagnosed within five minutes, reducing the average failure mitigation time by 20+ minutes. LogKG significantly reduces the amount of manual work required.

The remainder of this paper is organized as follows: In §2, we introduce the related works, NLP concepts, and challenges. In §2.3, we elaborate on LogKG's framework. Later in §4, we depict our experimental setup and results. Afterward, we describe the details of LogKG's deployment in §5 and conclude our paper in §6.

2 Related Work and Preliminaries

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

2.1 Related Work

2.1.1 Log-based Semantic Information Extraction

Some existing log anomaly detection works [1], [23]–[26] have been proposed to extract the semantic information of logs. LogAnomaly [1] obtains the semantic vector of a template by determining synonyms and antonyms. LogRobust [23] uses TF-IDF to give different word vectors different weights to obtain template vectors. [24] combines the transaction-level topic model for learning the embedding of logs.Log2vec [25] uses the semantic association to represent the relations by a graph embedding-based method.

In contrast to the methods described above for extracting semantic information from log entries, SLOGERT [27] and LEKG [26] model semantic information in logs using a KG. CoreNLP [28] is used by SLOGERT [27] to obtain the keywords information contained in log templates. It also employs regular expressions to determine the parameter type. LEKG [26] constructs triples from extracted semantic information using NER (Named Entity Recognition) [29] and background knowledge, and generates new triples using rule inference. When the log-based KG is combined with the background KG, the vast amount of information contained in background knowledge is utilized. None of these methods mine the semantics of logs' structured fields.

2.1.2 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 [16] [3] [15] [18] [30] [10] [20] [31] [22] [21]. Table 3 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) Execution Path Analysis. DeepLog [16] and LogFlash [15] use logs to construct log execution path and diagnose the root cause of failures. Since the above two methods do not explicitly give the root cause of a failure,they are inapplicable to our circumstances.

b) Failure-related Log Extraction. All three methods extract components from logs, compare normal and abnormal logs, and select failure-indicating logs to help diagnose failures. LOGAN [10] groups, parses, and aligns logs to detect normal patterns. After a failure, it calculates the log sequence divergence from a reference model and suggests a cause. Onion [8] extracts semantic information from the log, compares normal and anomalous logs, and selects the log that indicates the failure root cause to aid failure diagnosis.

FDiagV3 [9] uses computer cluster logs to diagnose failures. The three methods are inapplicable to our scenario because they aim to find logs that may indicate the root cause of failure.

c) Specific Type Root Cause Analysis. LADRA [17]can judge the time and location of faults. It uses training and detection in the form of real-time stream processing to determine the time of failure. A graph-based model is built to track the fine-grained request execution paths based on the migration probability to identify fault anomalies. It supports only four types of root causes, namely CPU, memory, network, and disk, which are inconsistent with our scenario.

d) Auxiliary Data Based Diagnosis. LogM [18] proposes a analysis and anomaly KG framework for real-world failure diagnosis. User actions help Ikeuchi et al. [20] find the cause. Nagara et al. [19] propose comparing system behavior logs in good and bad performance to diagnose performance failures. Log3C [21]uses cascading clustering to quickly group log sequences. Then, it finds significant issues that degrade KPIs. Wang et al. [22] suggest correlating log and KPI anomalies. These five works require a prior KG, user action data, or KPI data, which we can not obtain in a scenario.

e) Failure Type Determination. LogCluster [14] calculates log sequence vectors to represent log sequences and determines failure types by clustering. Yuan et al. [3] try to extract representation vectors from anomalous logs and use representation vectors to build a classifier to determine the failure type. But the clustering method provided by LogCluster doesn't use semantic information about the log itself. Yuan et al. [3] adopt supervised learning, which requires manual labels to train a classifier.

2.2 Preliminary

2.2.1 KG Construction and Embedding

LogKG utilizes the KG to fuse multi-field information. KG construction is the process of extracting structured information, and it includes three steps: entity extraction, relation extraction, attribute extraction, and entity alignment.

Entity Extraction is the process of finding named entities in text. This is usually done with algorithms like Yan et al. [32] and Strakova et al. [33] for named entity recognition (NER).

Relation Extraction aims to identify the relation of the given entity pair. Zeng et al. [34] propose an end-to-end model that can jointly extract relations from sentences. Entities are linked through relations to form a web-like knowledge structure.

Attribute Extraction aims to capture the attribute information of a specific entity from different information sources. Jiang et al. [35] can mine meta patterns in a massive corpus and can find high-quality attribute description statements as attribute values in the scenario of attribute extraction. we can treat entities with different label information as different entities, so the LogKG doesn't contain attribute extraction.

Entity Alignment aims at linking different entities with the same meaning. Some of the extracted entity names may differ, but they refer to the same entity, in which case they should be aligned. Zhu et al. [36] propose a novel Relation-aware Neighborhood Matching model named RNM for it.

KGE embeds the components of a KG, including entities and relations, into continuous vector spaces [37]. It can be categorized into two groups: translational distance models and semantic matching models.

TABLE 4: 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

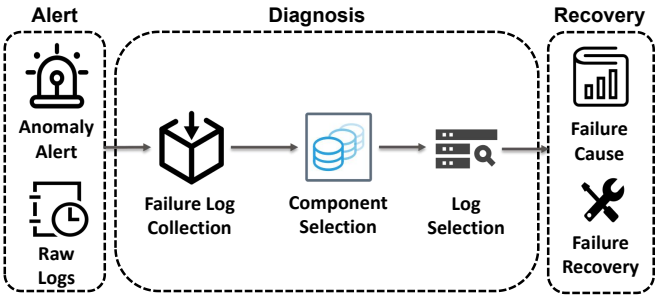


Fig. 2: The workflow of the manual failure diagnosis.

Translational Distance Models exploit distance-based scoring functions. They measure the plausibility of a fact as the distance between two entities, after a translation by the relation [37]. Relations are modeled in TransE [38] as translations acting on low-dimensional embeddings of entities, and both entities and relations are represented as vectors in the same space. KG2E [39] uses multivariate Gaussian random vectors to depict entities and relations. All connections are interpreted as complex vector rotations from one entity to another by RotatE [40].

Semantic Matching Models uses similarity-based scoring functions. Matching latent semantics of entities and relations in vector space representations [37] determines fact plausibility. RESCAL [41] assigns latent semantics to each entity. Each matrix represents latent factor pairwise interactions. DistMult [42] uses neural-embedding to learn entity-relation representations. Calculation and cost are simple.

2.2.2 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) [43] extracts semantic triples to represent the information in raw text, based on different NLP toolkits. LogKG uses it to extract semantic triples from log templates to represent events in logs.

Stanford CoreNLP is an extensible pipeline provides core natural language analysis [28]. It integrates various functions like NER, POS (part-of-speech), and OpenIE. LogKG performs triple extraction and keyword tagging based on CoreNLP to extract the information in the content field fully.

Common Event Expression (CEE) is a toolkit that offers a detailed taxonomy of general events. It improves the ability of users to interpret and analyze events [44] effectively. LogKG annotates the keywords in logs using CEE.

2.2.3 Manual Failure Diagnosis

Figure 2 shows the manual failure diagnosis workflow. After receiving a performance issue, operators will establish the failure's influence scope. Then, they will identify failure-related logs and determine the failure's root cause based on experience. Important are the failure-related logs. They will directly affect the results of the failure diagnosis. It is necessary to use the possible relations between logs to locate them.

In Table 1, we can see that a log has multiple fields representing different information. Log relations are contained in the content field and other fields. For example, different logs may share the same task ID or IP address. As mentioned in §1 and Figure 1, Operators cannot capture these relations when only the content field is used.

To reduce the workload, operators usually specify rules while searching a large number of logs. Even so, there are some disadvantages. On the one hand, labor-intensive to manually maintain these rules. On the other hand, it is error-prone to diagnose the root cause of failures based on these rules.

2.3 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 4, a log template can represent multiple events. For example, Table 4 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 the different events in a log need to be extracted as different event entities.

CH.2. Entity Alignment. Entity alignment aims to merge entities from different sources but semantically represent the same real-world object [45]. Without entity alignment, some entities with underlying relations cannot build associations with each other. For instance, in Table 4, "instance is VAR1" in the first template and "VAR1 is instance name" in the second template represent the same event. Intuitively, the first and second templates should build associations through the same event entity. However, existing methods cannot align the event entities 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 with TF-IDF [46]. However, unlike natural language, logs contain much noise. E.g., logs generated by scheduled tasks and normal business activities are usually irrelevant to failures and may interfere with failure diagnosis. The native NLP methods are unsuitable for obtaining failure-oriented log representations due to noise.

3 LogKG

3.1 The Framework of LogKG

LogKG has two parts: offline training and online diagnosis in Figure 3. When extracting entities, it not only uses information in the structured data but also extracts keywords and events from the unstructured texts. It aligns entities from different fields and fuses the multi-field information through a KG. It uses KGE to generate the representation vector of each failure. Then, it aggregates these failure cases by clustering. Operators label the root causes of typical cases for each cluster. After a failure occurs, LogKG converts the real-time logs into a representation vector and diagnoses the root cause based on the trained model.

Figure 4 shows the KG construction process, where the main steps are entity extraction and entity alignment. LogKG extracts triples from log templates in entity extraction, addressing the first challenge (*i.e.*, entity extraction). In entity alignment, LogKG aligns events through semantics, addressing the second challenge (*i.e.*, entity alignment). Then it builds a KG to fuse the aligned entities. It gets the representations of logs through KGE. Our proposed FOLR combines the local features of a failure and the global features of all failures to identify the failure-relevant logs. In this way, it obtains the failure-oriented representation vector for each failure, solving the third challenge (*i.e.*, failure-oriented log representation).

3.2 Entity Extraction

A log can be divided into two main parts: structured data and a template in Figure 4. Based on these two parts, we can extract three types of entities, i.e., structured entities, keywords, and event triples. Structured entities can be extracted directly. LogKG extracts keywords and event triples.

3.2.1 Keyword Extraction

Log templates contain keywords related to components, states, and tasks in the system, such as "MySQL", "DataBase", "AMQP", and "Error". These keywords are important for failure diagnosis. So they need to be extracted. Specifically, LogKG utilizes CoreNLP [28] to select keywords from log templates based on part-of-speech tagging.

3.2.2 Event Extraction

As mentioned, a log can contain multiple events. To efficiently extract the multiple events of each log. LogKG performs information extraction on templates, extracting event triples contained in them. Specifically, it combines Rule Extraction (RE) and OpenIE [47] to extract the triples.

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

OpenIE for Semantic Triples. OpenIE [43] is often used to extract semantic triples. LogKG utilizes it to extract semantic triples from logs, representing the multiple events in each log. There has been substantial progress in OpenIE approaches since it was proposed by Banko et al. [43]. 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", and "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. So each event is denoted as an entity through a triple.

3.3 Entity Alignment

Entity alignment aims to identify entities from different sources but semantically represent the same real-world object. We need to align these entities and merge the aligned

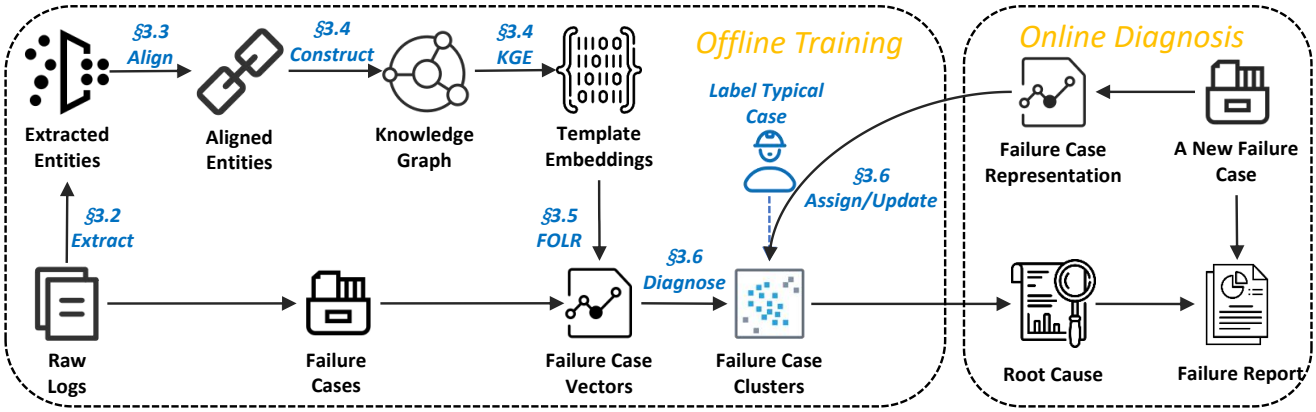


Fig. 3: The framework of LogKG

entities. In this way, we can build associations for the entities with underlying relations by the aligned entities. LogKG aligns keywords and events as follows.

3.3.1 Keyword Alignment

For keyword entities, LogKG aligns them by integrating NLP tools and domain knowledge. Specifically, it semantically annotates keyword entities based on the CEE library [44] and maps meaningful keywords into concept entities. However, the concepts contained in CEE are incomplete, so we extend the concepts of the CEE library based on domain knowledge. For example, the keyword "AMQP" can be annotated as "message middleware".

3.3.2 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 in syntax. Table 4 shows that 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. To associate these context-related log templates, we need to build associations for these templates using event entities. Therefore, 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 BERT [13] based pre-trained language models have achieved superior performance in recent years. We choose a pre-trained language model [48] 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 [49], hierarchical clustering [50], OPTICS [51], or others.

3.4 Knowledge Graph Construction and Embedding

3.4.1 Knowledge Graph Construction

LogKG requires storing large-scale unstructured data and displaying multi-field information association relations,

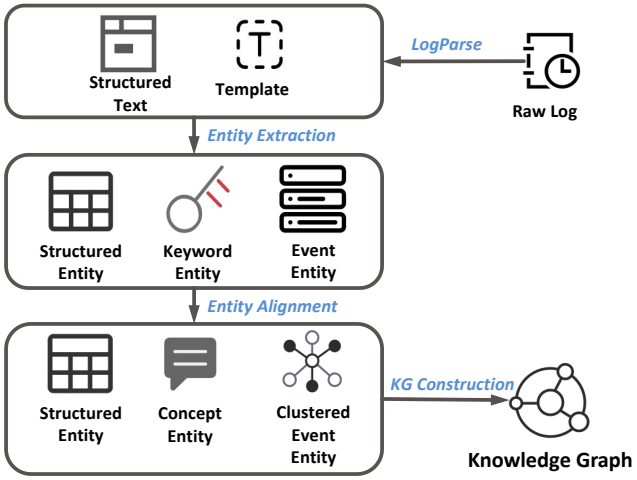


Fig. 4: The KG construction framework of LogKG

so Neo4j [52], which is based on graph structure storage, is applied. It has excellent performance for large-scale data queries and flexible unstructured data storage. It includes a Neo4j data browser to execute CQL (Cassandra Query Language) instructions which is a user-friendly interface.

After entity extraction and entity alignment, LogKG constructs a KG based on the explicit relations of different types of entities. Since logs contain limited information about the extracted entities, we cannot obtain the data properties of the entities. An entity's name represents the type and object properties of the entity. Figure 5 shows a subgraph of KG constructured by LogKG with different node types. We can see LogKG automatically associates the structured entities with different logs, e.g., "RequestID", "Template", "Level", "PID", "Component", "CEE". Logs are also mapped to log templates, which are associated with event entities and keyword entities, e.g., a log template is associated with "Event" and "CEE".

In this way, different entities extracted from multiple log fields are associated. We focus on the log templates and establish direct or indirect relations between the aligned entities and the log templates. These relations express the correlation between the semantics of different log templates.

The KG constructed by LogKG contains several types of entities and relations. As shown in Figure 5, the entity types include "Log", "RequestID", "PID", "Level", "Component",

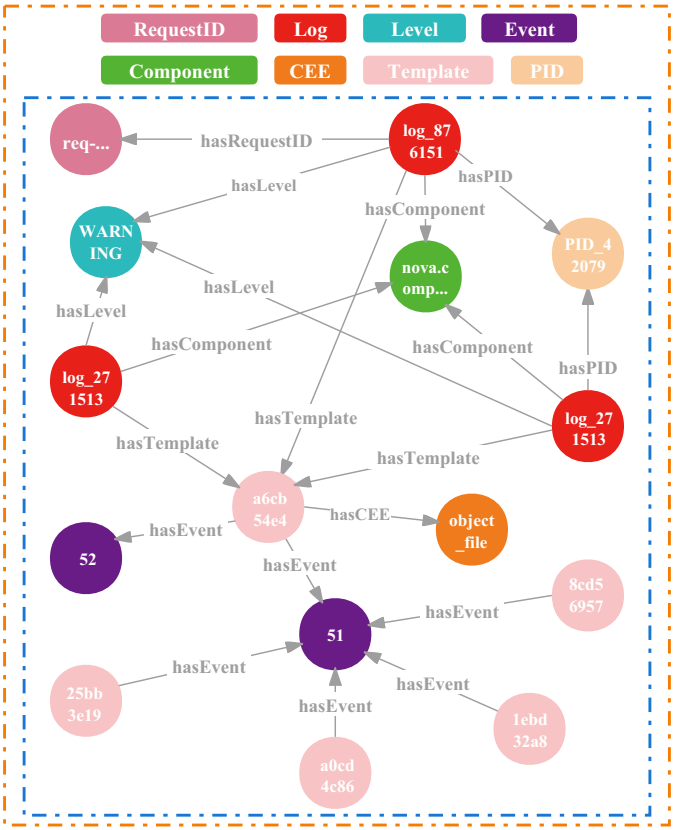


Fig. 5: A subgraph of an example KG constructed by LogKG.

"CEE", "Template", and "Event". Some of these entity types correspond to the structured fields in logs. Therefore, logs from different services may have different types of entities. Different entities usually have different relations. For example, the relations between the "Template" entities and the "Event" entities are "hasEvent".

3.4.2 *Knowledge Graph Embedding (KGE)*

We need to embed the logs into a vector space to diagnose failures based on logs. Based on the intuition that different logs corresponding to different types of failures should be easily distinguishable in a vector space, we need to make related logs closer and irrelevant logs farther apart in the vector space. Regular embedding methods can only embed the unstructured content field in the logs. They cannot leverage the multi-filed information in the logs, making it very difficult to identify failure-related logs in the vector space.

KGE embeds the components of a KG, including entities and relations, into continuous vector spaces [37]. It can utilize the multi-field information fused in the KG and calculate the semantic relation between entities in the vector space [53]. Therefore, we apply it to embed logs. While training the KGE model, we take the processed triplets representing the KG as input. We convert each entity and relation to a single identifier and represent triples as tuples of three identifiers. By taking the processed triples as the input, we can train a KGE model that maps entities and relations to the vector space.

Here, we adopt RotatE [40] as the KGE model. RotatE is a model with superior relation inference performance. It can describe the potential relation between various entities in 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 each log is mapped to a specific template, entities associated with a log also build an underlying relation with a template. In this way, we can transform the log sequence during a failure into a representation vector sequence of templates.

3.5 Failure-Oriented Log Representations

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 each failure. The time window length of collecting logs for each failure is ω, which varies with different types of services. A time window length ω indicates we collect logs of $\omega/2$ minutes before and after the time when a failure occurs. In our scenario, we set $\omega = 20$ minutes. Then, LogKG can map these logs to log templates to constitute a template set.

TF-IDF [54] is a term weighting technique in information retrieval [14]. Motivated by it, we propose Failure-Oriented Log Representation (FOLR). According to our observation, logs frequently appearing in all failure cases are usually irrelevant to failures. For each template, LogKG thus calculates its IFF (Inverse Failure Frequency) values. IFF represents the frequency of each template in all failure cases. Templates that appear more frequently in all failures usually have lower IFF values. They are usually irrelevant to a specific failure. In order to filter the noisy templates, LogKG ignores the log templates with lower IFF values. Moreover, for a failure-related log, its template's occurrence frequency differs in different types of failures. Therefore, for each template during a failure, LogKG calculates its TF (Template Frequency). We calculate the frequency of the occurrence of each template corresponding to a failure as the TF value. Then, LogKG uses TF-IFF to obtain a weighted sum of the representation vectors of the log templates.

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

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

where N is the total number of failures, n_t denotes the number of failures where template t appears, and θ_{iff} denotes the threshold of IFF. The TF of template $t(t \in T)$ of the i-th failure f_i is calculated as:

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

where $n_{t,i}$ is the number of times template t appears during failure f_i, n_i denotes the number of logs during f_i. By calculating TF and IFF, LogKG can obtain the representation vector V_i of failure f_i using weighted summation:

$$V_i = \sum_{j=1}^{m} w_{tf}(t_j, f_i) \times w_{iff}(t_j) \cdot e_j \quad (3)$$

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 obtain each failure case's representation vector for diagnosis.

3.6 Diagnosis

Failure diagnosis, which aims to identify the root cause of a failure, is usually formalized as a classification task. 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 clustering result.

3.6.1 Offline Training

To use the multi-field information fused by the KG 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 each failure. Subsequently, LogKG groups the representation vectors of failures into different clusters, and operators label the root cause of the typical failure for each cluster.

Failure Clustering. We employ a density-based clustering method to cluster the representation vectors of failures. Both DBSCAN [49] and OPTICS [51] are widely used density-based clustering algorithms. After analyzing the representation vectors of historical failures, we find that the densities of different failure types are pretty distinct. Therefore, we adopted the OPTICS clustering algorithm since our scenario satisfies its assumption and it is insensitive to hyperparameters. Here, we set the parameter MS, i.e., the minimum number of samples per cluster, to 3.

Failure Diagnosis. After clustering, operators select each cluster's most representative failure. 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 clusters. The root causes include network congestion, excessive memory usage, software bug, etc. In this way, we can reduce the labeling effort because the number of failure clusters (ten in our scenario) is much smaller than that of failures.

3.6.2 Online Diagnosis

LogKG is triggered when a service failure is detected at τ, after that It collects logs in the following closed intervals $[\tau - w/2, \tau + w/2]$ and calculates the failure representation vector as in the offline training stage. It assigns this failure to a failure cluster by calculating the distance between the calculated vector and the failure representation vectors of different clusters of failures. Some new failure types often appear, so the failure diagnosis model needs to be updated. For failures quite different from the existing ones, LogKG will update the clusters to adapt to the new failure types.

4 Evaluation

4.1 Experiment Design

4.1.1 Datasets

We conduct experiments over the GAIA dataset [5], 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 for the two datasets is listed in Table 5.

TABLE 5: 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

TABLE 6: Root Cause Of The Failures In GAIA Dataset

Failure type	Root cause
Login failure	Network congestion
Memory anomaly	Excessive memory usage
File not found	Misconfiguration
Access permission denied	Software bug

GAIA: The GAIA dataset is generated by simulating real-world microservice failures. It contains the records of all failure injections, including the timestamp, location, and all service logs of each failure. Specifically, 13,554,024 logs are related to the 1,083 failures, which can be classified into four types. Table 6 lists the failure types and root causes labeled by the operators who inject failures.

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 [55]. 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 [55], 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. We collected the 1,461,006 logs related to all randomly selected 93 failure cases in 24 days, which can be classified into six categories. Table 7 lists the failure types and root causes labeled 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.

4.1.2 Baselines

We compare LogKG with two failure diagnosis algorithms: unsupervised algorithm LogCluster [14] and supervised algorithm Cloud19 [3]. The parameters of these methods are all set best for accuracy. Specifically, a hierarchical clustering model is used to implement LogCluster, and we set its distance threshold to 0.5. For Cloud19, we choose Random Forest as the classification model and set the number of trees in the forest to 9.

4.1.3 Experimental Setup

We conduct all the experiments on the Jiutian platform provided by CMCC. It provides GPU cloud services with TeslaV100, 16-core CPU, and 32GB DRAM 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).

TABLE 7: Root Cause Of The Failures In CMCC Dataset

Failure type	Root cause
AMQP server unreachable	Network link down
Mysql lost connection Nova-conductor lost connection	Software bug
Computing node down	Hardware crash
Flavor disk too small Linuxbridge-agent anomalies	Misconfiguration

4.1.4 Evaluation Metrics

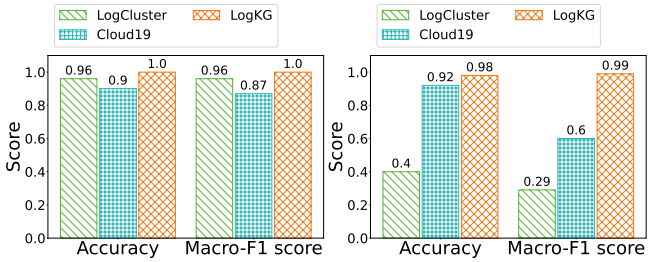
In our case, failure diagnosis can be viewed as a multiclassification problem. To measure the effectiveness of LogKG, we use accuracy and Macro-F1 score. Accuracy is the percentage of failure cases classified into the same category as the ground truth to all failure cases. To get a Macro-F1 score, we 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. For all failure cases, the recall is the percentage of failure cases with both the classification result and the 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 scores of all failure categories.

4.2 Evaluation of The Overall Performance

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

Figure 6 shows the comparison results of LogKG and the two baseline methods on the CMCC and 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 many 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 KG. Therefore, our method achieves superior failure diagnosis results than the two baseline methods.



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

Fig. 6: The effectiveness of different methods on the two datasets

4.3 Ablation Study

We conduct ablation experiments on the two datasets to evaluate the effectiveness of the two compulsory modules in LogKG: KG (knowledge graph) and FOLR (failure-Oriented log representations). Figure 7 shows the results of the ablation experiments on the CMCC and GAIA datasets.

4.3.1 KGE

In our proposed method, we build a KG with multi-field data and use KGE to get the embedding of each log template, and 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:

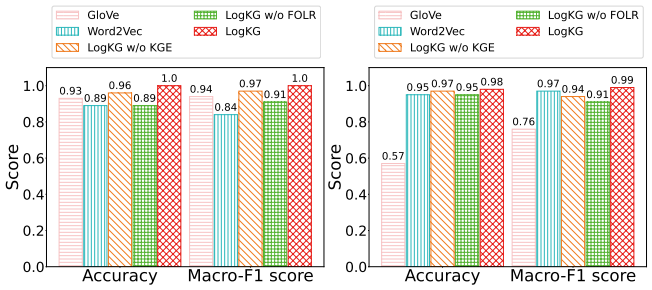
Removing KGE. We remove KGE from LogKG and use the one-hot vector to embed each log template. More specifically, suppose there are n log templates with indices from 1 to n in the whole dataset, then we use an n-dimensional vector to represent the i-th log template where the i-th bit in the vector is 1, and all other bits are 0. By doing so, we get the embedding of each log template. The performance of this method is displayed in Figure 7 as "LogKG w/o KGE".

Replacing KGE. 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 [56] and Word2Vec [57], 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 7, respectively.

From Figure 7, we can observe that KGE indeed improves the performance of LogKG. For example, LogKG without KGE achieves lower accuracy and 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 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 relatively well. LogKG extracts entities from logs and builds a KG. It not only preserves the semantic information of logs but also mines the relation between logs at a smaller granularity, which better adapts to the characteristics of logs. Thus we can conclude LogKG can achieve better accuracy with KGE.

4.3.2 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. In a real production environment, there are many noise logs during each failure,



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

Fig. 7: The effectiveness of LogKG when removing KGE, FOLR, or replacing KGE with different methods on the two datasets

easily leading to inaccurate classification. FOLR initially performs denoising by discarding parts of the logs with lower inverse failure frequencies, which are considered noisy data. It uses a threshold to decide which logs to discard.

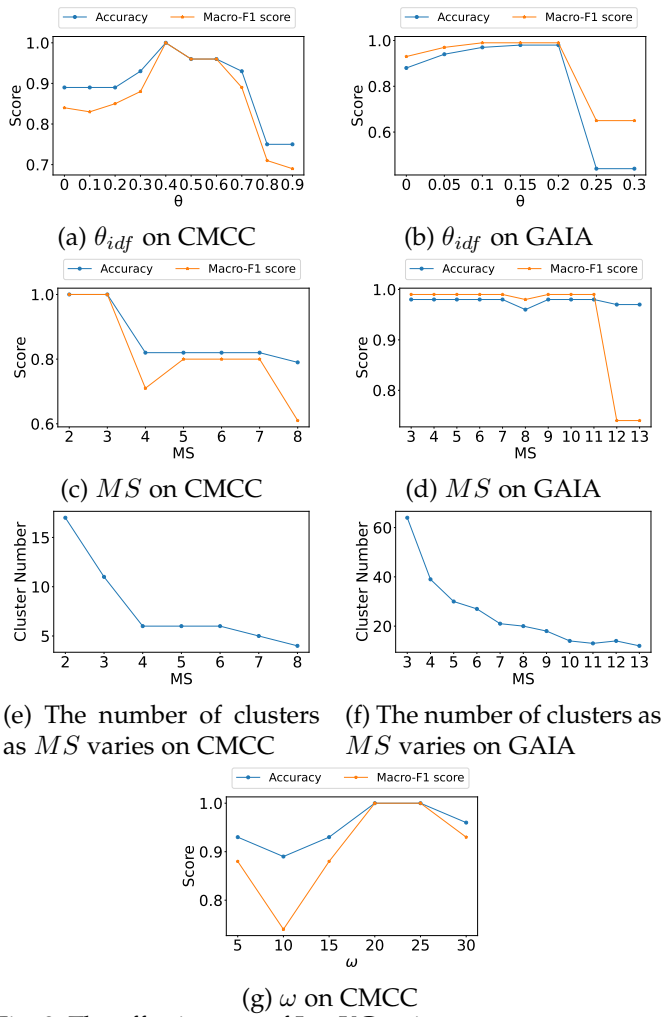
To verify the effectiveness of FOLR, we remove FOLR from LogKG and use the sum of the embeddings of log templates to get the embedding of each failure case directly. "LogKG w/o FOLR" in Figure 7 shows the performance of this method, and we can observe that LogKG without FOLR has lower accuracy and Macro-F1 score 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.

4.4 Evaluation of Hyper-Parameters

In this section, we evaluate the hyper-parameters of LogKG, which contains the threshold of FOLR (θ_{idf}), the minimum number of samples per cluster (MS) of OPTICS, and the time window length (ω), as shown in Figure 8. We evaluate θ_{idf} and MS on the two datasets and ω on the CMCC dataset because we use the duration of each failure, which can be obtained in the GAIA dataset, as a failure's corresponding time window. In addition, we discuss the effect of clustering algorithms and language models in the entity alignment, as well as the clustering algorithms used in the failure clustering on the two datasets.

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 in Figure 8(a) and Figure 8(b), we find 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, as shown in Figure 8(c) and Figure 8(d), LogKG achieves better when $MS < 4$ on the CMCC dataset and when $MS < 12$ on the GAIA dataset. The results 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 result. 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 8(e) and Figure 8(f), the number of clusters is negatively correlated with MS. It is less than 20 when MS



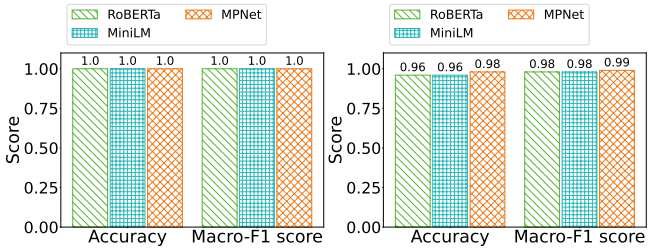
(a) θ_{idf} on CMCC (b) θ_{idf} on GAIA

(c) MS on CMCC (d) MS on GAIA

(e) The number of clusters as MS varies on CMCC (f) The number of clusters as MS varies on GAIA

(g) ω on CMCC

Fig. 8: The effectiveness of LogKG as its parameters vary on the two datasets

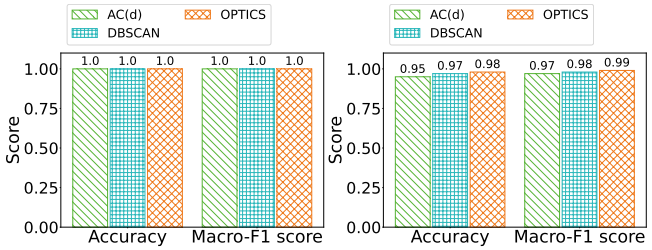


(a) Score on CMCC (b) Score on GAIA

Fig. 9: The effectiveness of LogKG as the language model of entity alignment varies on the two datasets

changes on the CMCC dataset and has no more drastic changes when $MS > 7$ on the GAIA dataset. We set the 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 8(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 that LogKG achieves the best performance when $\omega \in [20, 25]$. Therefore, we set $\omega = 20$. 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 large, more logs unrelated to the



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

Fig. 10: The effectiveness of LogKG as the clustering algorithm of entity alignment varies on the two datasets

failure will be introduced, making LogKG less effective.

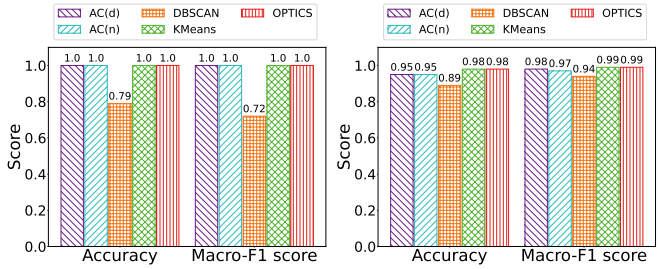
We discuss the several components used in entity alignment on the two datasets, including language models and clustering algorithms. We conduct experiments with three different language models: paraphrase-mpnet-base-v2(Mpnet), all-distilroberta-v1(Roberta), all-MiniLM-L12-v2(MiniLM) [58], and with three different clustering algorithms: OPTICS [51], Agglomerative Clustering [50] with specified distance threshold (AC (d)) and DBSCAN [49]. The results are shown in Figure 9 and Figure 10, respectively. All language models and clustering algorithms achieve 1.0 of both accuracy and Macro-F1 scores on the CMCC dataset. Although there are some minor variations, all the language models and clustering algorithms score above 0.95 in both accuracy and Macro-F1 scores on the GAIA dataset. The results show that varying language models and clustering methods in the entity alignment cause little impact on the effectiveness of LogKG. We choose Mpnet as the language model and OPTICS as the clustering method in entity alignment since they respectively performed the best in the GAIA dataset.

We also discuss the effect of different clustering algorithms in failure clustering on the two datasets. We conduct experiments with five different clustering algorithms: OPTICS, Agglomerative Clustering with specified distance threshold (AC (d)), Agglomerative Clustering with a specified number of clusters (AC (n)), DBSCAN and KMeans, and the results are shown in Figure 11(a) and Figure 11(b), respectively. Except for DBSCAN, other clustering algorithms all achieve 1.0 of both accuracy and Macro-F1 score on the CMCC dataset and achieve $0.95+$ of 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. We choose OPTICS as the clustering method in failure clustering since the experiments show that it and KMeans perform the best of the five algorithms.

4.5 Threats to Validity

4.5.1 Data Quality

LogKG consists of two parts: KG construction and failure diagnosis model training. The performance of the failure diagnosis model depends on the effect of KG 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 KG construction. In the face of complex templates or log formats, it's necessary to manually improve the quality of the log template. These additional workloads don't require human effort.



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

Fig. 11: The effectiveness of LogKG as its clustering algorithm varies on the two datasets

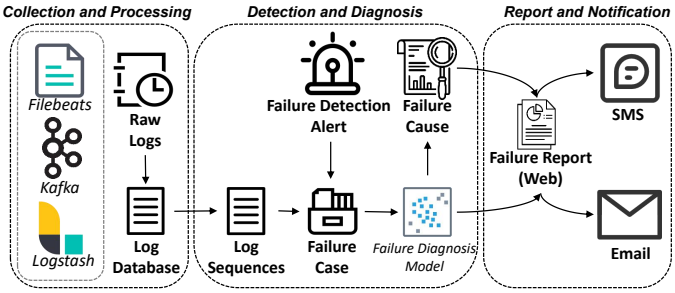


Fig. 12: The workflow of LogKG in the production environment of CMCC.

4.5.2 Scope of Logs during Failure

In our experiments, we set different ω for the GAIA and the CMCC datasets, respectively. For the GAIA dataset, since the failure duration 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 set $\omega = 20$. 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 logs during the failure, thereby affecting failure diagnosis performance. Since the amount of data in our experiments is still limited, verifying the impact of the above situations is impossible. We will verify it on more online large-scale services in future work.

5 Deployment & Case Study

We have deployed LogKG in CMCC to verify its performance in KG construction and failure diagnosis. Compared with manual failure diagnosis described in §2.2.3, operators don't need to do a lot of error-prone and inefficient log searching. LogKG shortens failure mitigation time and increases failure diagnosis accuracy.

5.1 Deployment in CMCC

5.1.1 Workflow

Figure 12 shows the workflow of LogKG in the production environment of CMCC, including three steps:

Step. 1: Collection and Processing. CMCC uses Filebeat [59] to aggregate log files from different service instances and writes log streams to Kafka [60]. Operators connect Kafka with Logstash [61], which preprocesses logs utilizing extensions and plugins. They match the logs to templates using the log parsing methods. The logs can be processed and stored in the database soon after they are generated.

Step. 2: Failure Diagnosis. When a failure is detected through failure detection methods, LogKG automatically collects logs before and after it. Subsequently, LogKG computes the representation vector of the failure based on these logs and uses the clustering model trained offline to diagnose the root cause of the failure.

Step. 3: Report and Notification After LogKG diagnoses the failure, the system will generate a failure report. It includes the possible root causes diagnosed by LogKG and suggestions for failure mitigation based on historical experience. Then, it will be displayed on the Web page and sent to the operators via SMS and email.

5.1.2 Performance

Specifically, we have deployed LogKG in Guangdong Mobile Communications Co., Ltd (GMCC). LogKG is responsible for diagnosing failures for more than 2000 service instances, which generate hundreds of millions of logs daily. During the five months of deployment, LogKG averagely processed 47 failures daily. Based on LogKG, the failure diagnosis can be completed within *five minutes*, and the average failure mitigation time has been significantly reduced by more than *20 minutes*.

5.2 Case Study

Here, we use two selected failure cases to compare the amount of work required using LogKG and manual keyword search, respectively. The outage of the MySQL component caused the first failure, and the other was due to the exception of the Nova Conductor component. As listed in Table 8, we count the number of logs that need to be investigated containing different keywords during the above two failure cases. As mentioned above, the support services such as MySQL and AMQP are used in OpenStack to provide functions such as database and communication. When the MySQL service failed, some services also failed because of it. For example, "Unexpected error while reporting service status" will be printed in other service logs. Moreover, Nova Conductor is a common component in OpenStack. Its exception caused a VM creation failure. It resulted in various logs in several components, such as "Policy check failed with credentials: ..." and "qemu-img failed to execute". Although the operators can search by keywords to find the failure information provided by these logs, it requires much manual work. For instance, they have to examine 2500+ or 200+ logs to find the logs indicating the root causes of the two failures.

LogKG significantly reduces manual effort. It automatically calculated the failure vector after an online production failure. The offline failure diagnosis model classified it into known root causes and generated a failure report. The whole process is completed without any manual effort.

TABLE 8: The Number Of Logs Containing Different Keywords During Two Real-Word Failure Cases

Failure Case	Kill	Fail	Error	Exception
Nova Conductor Error	161	60	4	4
MySQL Shutdown	322	15	811	1449

6 Conclusion

In this paper, we fuse the multi-field information in logs to automatically construct a knowledge graph, which can extract more information from logs. We use the knowledge graph information to train a failure diagnosis model and finally determine the cause of the failure. Extensive experiments using a simulation dataset and a real-word dataset have demonstrated the superior performance of LogKG and the importance of its key components. We take the first step to fuse the multi-field information in logs using a knowledge graph. In the future, we will verify LogKG's performance in more scenarios.

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] CloudWise-OpenSource, "Gaia-dataset," [EB/OL], https://github.com/CloudWise-OpenSource/GAIA-DataSet Accessed October 4, 2022.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] 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.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] 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.

[17] 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.

[18] 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.

[19] 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.

[20] 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.

[21] 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.

[22] 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.

[23] 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.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] B. Mohit, "Named entity recognition," in *Natural language processing of semitic languages*. Springer, 2014, pp. 221–245.

[30] 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.

[31] 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.

[32] H. Yan, T. Gui, J. Dai, Q. Guo, Z. Zhang, and X. Qiu, "A unified generative framework for various ner subtasks," *arXiv preprint arXiv:2106.01223*, 2021.

[33] J. Straková, M. Straka, and J. Hajič, "Neural architectures for nested ner through linearization," *arXiv preprint arXiv:1908.06926*, 2019.

[34] X. Zeng, D. Zeng, S. He, K. Liu, and J. Zhao, "Extracting relational facts by an end-to-end neural model with copy mechanism," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2018, pp. 506–514.

[35] M. Jiang, J. Shang, T. Cassidy, X. Ren, L. M. Kaplan, T. P. Hanratty, and J. Han, "Metapad: Meta pattern discovery from massive text corpora," in *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2017, pp. 877–886.

[36] Y. Zhu, H. Liu, Z. Wu, and Y. Du, "Relation-aware neighborhood matching model for entity alignment," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 35, no. 5, 2021, pp. 4749–4756.

[37] 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.

[38] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in Neural Information Processing Systems*, C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, Eds., vol. 26. Curran Associates, Inc., 2013. [Online]. Available: https://proceedings.neurips.cc/paper_files/paper/2013/file/1cecc7a77928ca8133fa24680a88d2f9-Paper.pdf

[39] S. He, K. Liu, G. Ji, and J. Zhao, "Learning to represent knowledge graphs with gaussian embedding," in *Proceedings of the 24th ACM international on conference on information and knowledge management*, 2015, pp. 623–632.

[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] M. Nickel, V. Tresp, H.-P. Kriegel *et al.*, "A three-way model for collective learning on multi-relational data." in *Icml*, vol. 11, no. 10.5555, 2011, pp. 3 104 482–3 104 584.

[42] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[43] M. Banko, M. J. Cafarella, S. Soderland, M. A. Broadhead, and O. Etzioni, "Open information extraction from the web," in *CACM*, 2008.

[44] "Mitre: Common event expression." Website, 2014, https://cee.mitre.org/.

[45] 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

[46] 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.

[47] 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.

[48] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," in *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, 2019, pp. 3982–3992.

[49] 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.

[50] 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.

[51] 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.

[52] D. Fernandes and J. Bernardino, "Graph databases comparison: Allegrograph, arangodb, infinitegraph, neo4j, and orientdb," in *7th International Conference on Data Science, Technology and Applications*, 2018.

[53] Y. Dai, S. Wang, N. N. Xiong, and W. Guo, "A survey on knowledge graph embedding: Approaches, applications and benchmarks," *Electronics*, vol. 9, no. 5, 2020. [Online]. Available: https://www.mdpi.com/2079-9292/9/5/750

[54] H. Schütze, C. D. Manning, and P. Raghavan, *Introduction to information retrieval*. Cambridge University Press Cambridge, 2008, vol. 39.

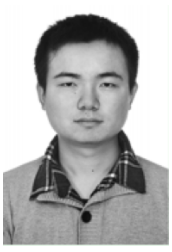
[55] T. Rosado and J. Bernardino, "An overview of openstack architecture," in *Proceedings of the 18th International Database Engineering & Applications Symposium*, 2014, pp. 366–367.
[56] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543.
[57] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.
[58] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," 2019.
[59] B. Elasticsearch, "Filebeat-lightweight shipper for logs (2020)," *URL https://www. elastic. co/products/beats/filebeat. Accessed*, pp. 02–12, 2021.
[60] 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.
[61] E. Stack, "Elasticsearch, logstash, kibana— elastic," *URL: https://www. elastic.co/what-is/elk-stack*, 2021.

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

Yuzhe Zhang received a B.S. degree in Software Engineering from Nankai University in 2020. He is currently an M.S. student at 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 management and NFV cloud network architecture.

Ting Xu received the M.E. degree in Software Engineering from Central South University in 2020. She is currently a Ph.D. student at the College of Software, Nankai University. Her research interests include anomaly detection and failure diagnosis. She is an IEEE Member.



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 at the College of Software, Nankai University. Her research interests include Artificial Intelligence, Software Engineering, etc.

Yongqian Sun received a 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 at the College of Software, Nankai University, Tianjin, China. His research interests include anomaly detection and root cause localization in service management.

Fangrui Guo received the M.E. degree in Software Engineering from Nankai University in 2020. Her research interests include anomaly detection and failure diagnosis.

Junyu Shen received the B.S. degree in Software Engineering from Nankai University in 2022. He is currently an M.S. student at the College of Software, Nankai University. His current research interest includes 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 of artificial intelligence.

Dan Pei received the B.E. and M.S. degrees 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 at 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.

Xiao Yang received M.S. in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2017. He is currently the project manager of the Innovation Research Institute of China Mobile. His current research interests include network management.

Li Yu received M.S. in communication engineering from Beijing University of Posts and Telecommunications, Beijing, China, in 2007. He is currently the vice president of the Innovation Research Institute of China Mobile (Zhejiang) His current research interests include mobile communication technology, network intelligence, and big data of communication networks.