LogSummary: Unstructured Log Summarization for Software Systems

Weibin Meng, Federico Zaiter, Yuzhe Zhang, Ying Liu, *Member, IEEE*, Shenglin Zhang, *Member, IEEE*, Shimin Tao, Yichen Zhu, Tao Han, Yongpeng Zhao, En Wang, *Member, IEEE*, Yuzhi Zhang, and Dan Pei, *Senior Member, IEEE*

***Abstract*—We propose LogSummary, an automatic, unsupervised end-to-end log summarization framework for software system maintenance in this work. LogSummary obtains the summarized triples of necessary logs for a given log sequence. It integrates a novel information extraction method that considers semantic information and domain knowledge with a new triple-ranking approach using the global knowledge learned from all logs. Given the lack of a publicly-available gold standard for log summarization, we have manually labeled the summaries of four open-source log datasets and made them publicly available. The evaluation of these datasets and the case studies on real-world logs demonstrate that LogSummary produces highly representative (average ROUGE F1 score of 0.741) summaries efficiently. We have packaged LogSummary into an open-source toolkit and hope it can be a standard baseline and benefit future log summarization works.**

***Index Terms*—AIOps, log analysis, log summarization.**

I. Introduction

With the continuous development of Internet applications, large-scale software systems are getting increasingly large and complex. A non-trivial software anomaly can impact the user experience of millions of users and lead to significant revenue loss [1]. Consequently, the reliability of software systems is of vital importance.

Manuscript received 17 July 2022; revised 6 December 2022; accepted 9 January 2023. Date of publication 16 January 2023; date of current version 9 October 2023. This work was supported by the National Key R&D Program of China under Grant No. 2021YFB0300104, the National Natural Science Foundation of China under Grant No. 62272249, 62072264, and 61902200, and the China Postdoctoral Science Foundation under Grant No. 2019M651015. The associate editor coordinating the review of this article and approving it for publication was D. Pezaros. *(Corresponding author: Shenglin Zhang.)*

Weibin Meng, Shimin Tao, Tao Han, and Yongpeng Zhao are with Huawei Inc., Beijing 100085, China (e-mail: mengweibin3@huawei.com; taoshimin@huawei.com; billow.han@huawei.com; zhaoyongpeng@huawei.com).

Federico Zaiter is with the Department of Artificial Intelligence and Big Data, Universidad ORT Uruguay, Montevideo, Uruguay (e-mail: federico.zaiter@fi365.ort.edu.uy).

Yuzhe Zhang is with the College of Software, Nankai University, Tianjin 300071, China (e-mail: zyzcs@mail.nankai.edu.cn).

Ying Liu, and Dan Pei are with the Department of Computer Science and Technology, Tsinghua University, Beijing 100084, China (e-mail: liuying@cernet.edu.cn; peidan@tsinghua.edu.cn).

Shenglin Zhang and Yuzhi Zhang are with the College of Software, Nankai University, Tianjin 300071, China, and also with the Haihe Laboratory of Information Technology Application Innovation, Tianjin 300450, China (e-mail: zhangsl@nankai.edu.cn; zyz@nankai.edu.cn).

Yichen Zhu is with Midea Group, Foshan 528311, China (e-mail: k.zhu@mail.utoronto.ca).

En Wang is with the Department of Computer Science and Technology, Jilin University, Changchun 130012, China (e-mail: wangen0310@126.com).

Digital Object Identifier 10.1109/TNSM.2023.3236994

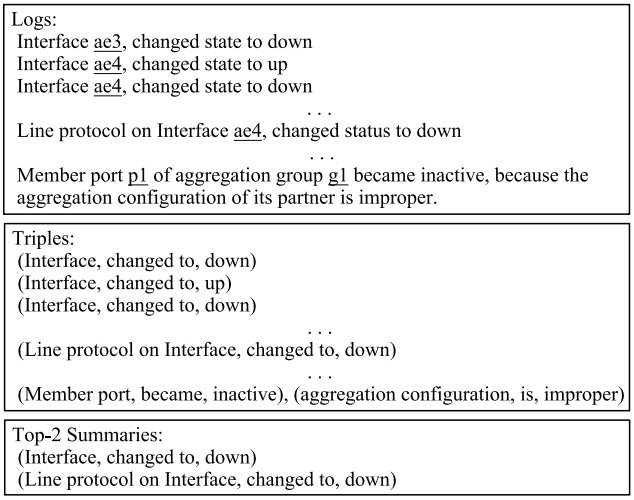

Fig. 1. The triples and top-2 summaries of an example log stream. We extract the triples from the logs and rank them based on semantic importance, then select the two most important triples as top-2 summaries.

Large-scale software systems usually generate logs (see the top half of Fig. 1), which describe a vast range of events observed by them and are often the only available data recording software runtime information. Therefore, many automatic log analysis approaches have been proposed for software system management [2], which can be classified into log compression methods (*e.g.,* [3]), log parsing methods (*e.g.,* [4], [5], [6]), anomaly detection methods (*e.g.,* [7], [8]), failure prediction methods (*e.g.,* [9]), and failure diagnosis methods (*e.g.,* [10], [11]), *etc.* Although these approaches help operators efficiently understand the status of software systems, they leave the burden of summarizing logs to operators. More specifically, after a failure is detected/predicted/diagnosed, operators still have to read the corresponding original logs (*i.e.,* a log sequence) to understand software-wide semantics [12]. It is because the existing automatic log analysis approaches, especially those for failure prediction or diagnosis purposes, are not accurate and general enough for every scenario. Before taking measures to mitigate or avoid failures, operators must ensure that a failure has occurred or will occur.

However, manual log summarization, or the rule (*e.g.,* regular expression rule) based log summarization, has become ineffective and inefficient for the following three reasons. (1) A large-scale software system is usually implemented and maintained by hundreds of developers and operators. The

developers or operators who investigate logs often have incomplete knowledge of the original logging purpose. (2) The volume of logs is proliferating, for instance, at a rate of about 50 gigabytes (around 120∼200 million lines) per hour [5]. The traditional way of log summarization, which largely relies on manual inspection or rule update, has become a labor-intensive and error-prone task. (3) With agile software development becoming increasingly popular, operators deploy software updates more frequently, leading to a large number of new types of logs being generated continuously. It is challenging for operators to timely comprehend these new types of logs. Although several works have been proposed for log compression [3], [13] or system interpretability through logs [14], [15], to the best of our knowledge, there is no previous work aiming to help operators efficiently and effectively summarize a given log sequence in an interpretable and readable manner.

When operators investigate logs, they usually care about three key pieces of information, *i.e.,* "entities", "events" and the "relation" between them. These three elements form a relation tuple ("entities", "events", "relation"), which we will call ***triple*** [16] in the following. Sometimes only two elements are needed for representation, and the first or third position of the triple may be blank. In most cases, a log contains one or more triples. These triples are easy to understand for operators because they keep both the semantics and syntax of the original log [17]. Suppose we can automatically select some important summarized triples in a given log sequence as ***log summarization*** and provide them to operators. In that case, they can gain a clear view of this log sequence. Triples that do not contain important semantic information should not appear in the log summarization. Otherwise, they will increase the workload of operators. As shown in Fig. 1, we select a part of the log sequence for display and list their corresponding summarized triples. For example, ("Interface", "changed to", "down") is the expected relational triple extracted from the first log, and both ("Member port", "became", "inactive") and ("aggregation configuration", "is", "improper") are the expected relational triples extracted from the last log. Considering the semantics importance and frequency, we choose the two most important triples as the top-2 summaries, and the detailed ranking policy will be introduced in Section V.

The original goal of logs, *i.e.,* "*logs are designed for operators to read*", motivates us to apply natural language processing (NLP) methods to summarize logs. Compared with the NLP log summarization methods, most non-NLP methods process logs with the help of statistical information, such as word frequency in logs, and do not fully explore the semantic information. A typical non-NLP method is log template extraction, which can filter out parameter information and extract the fixed part of logs. However, this method cannot help operators quickly understand the log semantics in many cases. As shown in Fig. 2, the raw log has 56 words, and its corresponding template has 26 words. However, the semantic triples extracted by an NLP method has only 5 words, which greatly saves operators' time for understanding this log. When the log is long, a log template extraction method neither significantly shortens the length of the text nor gives clear semantic information in the log. A large number of works have been proposed for text summarization in the NLP domain [18]. However, the four following challenges lie in applying existing NLP methods for log summarization.

Log:
4643092 (5438708) iar 0x00544eb8 , dear 0x01fc5640 (543 RAS KERNEL INFO Kernel detected 5438714 integer alignment exceptions (5438707) iar 0x00544ea8 , dear 0x01fc5620 (5438708) iar 0x00544eb8 , dear 0x01fc5640 (5438709) iar 0x00544ea8 , dear 0x01fc5660 (5438710) iar 0x00544eb8 , dear 0x01fc5680 (5438711) iar 0x00544ee0 , dear 0x01fc56a0 (5438712) iar 0x00544ef0 , dear 0x01fc56c0 (5438713) iar 0x00544ee0 , dear 0x01fc56e0 (5438714) iar 0x00544ef0 , dear 0x01fc5700

Template:
* (*) iar * , dear * (* RAS KERNEL INFO Kernel detected * integer alignment exceptions (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear * (*) iar * , dear *

Triple:
(Kernel, detected, integer alignment exceptions)

Fig. 2. Log template and semantic triple.

(1) It is difficult for the existing NLP tools to extract the expected triples from logs accurately because logs contain not only normal words, but also domain-specific symbols, and the syntax of a large portion of logs significantly differs from normal sentences.

(2) Large-scale software systems can generate a massive number of logs in a short period. Typically, operators usually investigate all the logs of some period (say one hour before a failure) to obtain the summary of these logs [19]. Applying existing NLP tools to extract triples for each newly generated log is computationally inefficient (see Table VI).

(3) Existing NLP methods summarize texts based on the order of sentences (logs) [20]. However, operators expect to read the summarization of *important* logs first for a given log sequence. Therefore, there is a vast gap between operators' expectations and the summarization generated by existing NLP methods.

(4) Applying NLP methods to learn a text summarization model usually needs a large-scale training set [18]. However, to the best of our knowledge, although there are publicly available log datasets [5], there is no publicly available dataset for log summarization yet.

To address the above challenges, we propose LogSummary, an automatic, unsupervised end-to-end log summarization framework for software systems. The goal of LogSummary is to obtain the summarized triples of important logs for a given log sequence, which takes both semantic information and domain knowledge into consideration. The contributions of this paper are summarized as follows.

(1) We propose LogSummary, a new framework to perform log summarization for software systems. For a given log sequence, LogSummary obtains the summarized triples of important logs. The summary preserves the important information of this log sequence and is easy to understand. The implementation of LogSummary is available online.[1]

[1]https://github.com/LogSummary/code-and-datasets

(2) We propose a new domain-specific and efficient information extraction approach called LogIE (Log Information Extraction). It accurately extracts the expected triples for each log by integrating it with domain knowledge (addressing challenge 1) and achieving efficient information extraction by combining it with a log template (addressing challenge 2).

(3) We propose a new simple yet effective method to rank the triples generated by LogIE. It ranks triples according to the global knowledge learned from all logs rather than each triple's local information. In this way, LogSummary accurately obtains the triples of *important* logs expected by operators (addressing challenge 3).

(4) Given the lack of a publicly-available gold standard for log summarization, we manually labeled reference summaries for four existing open-source log datasets and made them available on github.[2] We believe that the availability of a summary gold standard would benefit future research and facilitate the adoption of automated log summarization. Extensive experiments based on these summaries demonstrate that LogSummary generates highly representative (average ROUGE F1 score of 0.741) summaries, and LogIE is more accurate and much more efficient than baseline methods.

The rest of the paper is organized as follows: We introduce the related work in Section II, discuss the background in Section III and highlight the challenges in Section IV. In Section V, we propose our approach. The evaluation is shown in Section VI. In Section VII, we discuss the practicability of LogSummary. Lastly, we conclude our work in Section VIII.

II. Related Work

Different log summarization approaches have been proposed for purposes different from ours. At the same time, different methods have been used for similar purposes to enhance the interpretability of the logs for troubleshooting.

Gentili et al. [13] proposed an approach to leverage a taxonomy graph over the events presented by the logs in order to reduce the resulting raw data size to keep the required space to store the data manageable and improve the performance and system load for log analysis. Gunter et al. [19] presented an approach for log summarization to decrease the load on the system for logging. They do not focus on the human interpretability of the logs but rather on reducing system workload for downstream tasks. Their main evaluation metric is keeping the same performance of autonomous anomaly detection despite the compressed logs. Shang and Syer focus on understanding software logs [21] and examining the stability of logs [22]. Huo et al. [23] proposed an approach to capture message-level and instance-level semantics from logs, and they focus on the relationship between concepts and instances in logs. He et al. [24] presented a method to identify impactful service problems by using log analysis. They pay more attention to visualization rather than readability. Le and Zhang [25] used an NLP method to extract the semantic meaning of raw log messages, which they then used to detect anomalies. They do not focus on semantic importance ranking and readability either.

Satpathi et al. [15] proposed a closely related work in our scenario. Their focus is different from ours as they aim to mine the distribution of messages for each anomalous event and their occurrences in the logs, getting an event signature that represents it formed by keywords. Nonetheless, they use an approach that resembles log summarization by first adopting their proposed change-point detection to aggregate all the events from the given distribution, followed by summarization utilizing LDA [26]. However, LDA has proven ineffective for short text summarization despite aggregation or previous clustering. Additionally, they do not consider parameters from the logs that are key to understanding an event after summarization and helping operators with their troubleshooting. Chen et al. [27] proposed an improved word mover's distance to measure the distance between two log samples, and Otomo et al. [28] proposed an approach to obtain a numerical topic-distributed representation of each log. Both two methods use LDA to obtain the log topic distribution information, and they also have the above-mentioned drawbacks and pay more attention to the categories of logs rather than summarization.

III. Background

Log Parsing: Log parsing usually serves as the first step towards automated log analysis, and it can pre-process the unstructured part of the log. The most popular log parsing approach is automatic template extraction [4], [5], [29], [30], which extracts constant fields (templates) from logs. For example, "Interface *, changed state to down" is the template of the first and fifth logs in Fig. 1, and traditional log parsing methods can extract templates from historical logs automatically. However, operators continuously conduct software/firmware upgrades to introduce new features, fix bugs, or improve performance. These upgrades usually generate new types of logs required to update templates online [22]. Therefore, we utilize online log parsing methods (*e.g.,* LogParse [4]) to extract templates in this work. Online log parsing methods can extract and learn templates online without retraining their model. Note that log parsing is the precursor to information extraction and log summarization. We generate the log template through log parsing and then use information extraction to get triples from the template. Finally, we generate the log summarization from the triples.

Word Embedding for Logs: Logs are designed to facilitate user readability. Consequently, the constant parts of logs are defined in a human-readable manner by developers. Many methods (*e.g.,* word2vec [31]) thus use natural language process (NLP) methods to represent words. However, these methods cannot represent domain-specific words accurately. For example, "down" and "up" in Fig. 1 are antonyms but have similar contexts. Besides, system upgrades usually generate new types of logs with unseen words [32] (*e.g.,* "Vlan-interface" in Fig. 1), which poses a challenge for generating distributed representations of words in logs. For this reason, we adopt Log2Vec [32] to represent the

[2]https://github.com/LogSummary/code-and-datasets

words of logs. Log2Vec combines a log-specific word embedding method to accurately extract the semantic information of logs with an out-of-vocabulary (OOV) word processor to embed unseen words into a distributed representation at runtime.

Information Extraction: Information extraction retrieves relational triples from unstructured text. It is usually done in the form of triples for binary relations, relating two arguments by a predicate or relation for each relation in a given sentence, *i.e.,* $(argument_1, relation_{1,2}, argument_2)$ [16]. Traditional information extraction approaches rely on predefining a limited set of target relations and hand-crafted patterns. For this, they would adopt Named Entity Recognizers and dependency parsers to target a specific domain. These approaches would then require manual effort to be repurposed and applied to a different domain. To address the challenge of scalability for performing Web information extraction, Banko et al. [16] introduced Open Information Extraction (OpenIE). They presented a new information extraction paradigm that would allow extracting relations without defining the number or type of relations in advance.

IV. Challenge and Overview

A. Empirical Study

When a failure occurs in a system, operators can use log summaries to figure out why the failure occurred as quickly as possible. In the traditional method of manually extracting summaries, operators observe the logs to extract the summaries. For example, an experienced operator can summarise from "Line protocol on Interface ae4, changed state to down." within about 3 seconds that "Line protocol changed to down" is the main event described in the log. As log complexity increases, this process can take a little longer, as each log may contain more than one event. Meanwhile, the system may generate thousands of logs per minute, making manual extraction of log summaries infeasible in the production environment. In the rule-based summary extraction approach, a rule must be defined for each log type to achieve good results. In the case of a new type of log being generated in the system, for example, "Image 1.jpg at server A in use" contains the main event "Image is in use", but there is no way to extract a summary based on rules without adding a specific rule. The requirement to read and understand logs quickly in software systems motivates the design and implementation of LogSummary.

B. Design Challenges

Log data is an important data source recording system states and significant events at runtime. Thus, it is intuitive for operators to observe the system's status and inspect any potential anomalous events using logs. A log is usually printed by logging statements (*e.g.,* printf(), logger.info()) in the source code, which developers predefine. Typically, the predefined part of a log is human-readable. Therefore, solving log summarization problems using NLP tools seems promising. However, directly applying existing NLP approaches for log summarization faces several challenges as follows.

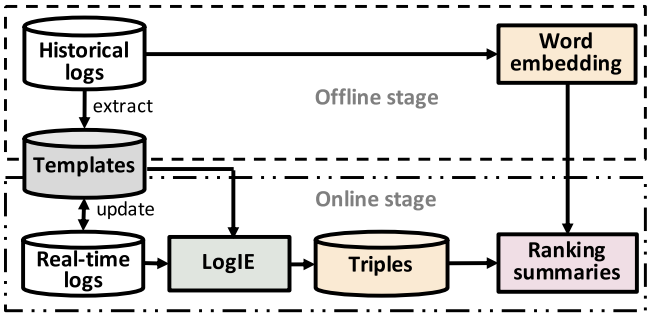

Fig. 3. Framework of LogSummary.

Domain-specific symbols and grammar: Logs contain many domain-specific symbols, and their grammar may significantly differ from normal sentences. Existing NLP tools, which are typically designed for normal sentences, cannot get accurate summaries for them. For example, entity-value pairs are valuable and structured information that should be extracted from unstructured logs. However, existing NLP tools cannot extract them directly because they may be separated by an equal "=" or a colon ":" symbol. Besides, some entity-value pairs are hidden in word combinations. For instance, when NLP tools process the first log in Fig. 1, it may treat "Interface ae3" as a whole, while "ae3" is a value for the entity of "Interface".

High summarization efficiency requirement: After a failure is detected or predicted, the operator hopes to quickly obtain the summary of a collection of logs in some period (*e.g.,* one hour before a detected failure) to figure out what happens on the software system. However, the software system can generate a large number of logs during this period. For example, one program execution in the HDFS system generates 288,775 logs per hour [7]. On the other hand, existing NLP methods typically get the summarized triples one sentence (log) by one sentence (log), and their efficiency cannot satisfy the requirement of operators (see Table VI for more details).

Obtaining the summarized triples of important logs: Typically, logs are generated in the order of program execution, and they contain redundancy and repetition. When operators inspect a collection of logs triggered by a failure detection or prediction, they want to obtain the triples of those *important* logs first. However, existing NLP approaches usually generate summaries according to the order of sentences (logs) in the original text (log sequence). In Fig. 8, for example, a state-of-the-art NLP method generates summaries by compressing original logs instead of generating triples of the expected important logs. Consequently, these approaches cannot satisfy the expectation of operators.

C. Overview of LogSummary

In this paper, we design LogSummary (as shown in Fig. 3) to summarize logs in software systems and help operators read/understand logs faster. LogSummary is an automatic log summarization approach that takes both semantic information and domain knowledge into consideration, which differs from existing methods and achieves a good result. We decompose the LogSummary into the offline training and online

summarization stage, involving word embedding and log templates, which we will describe in detail. LogSummary needs to obtain both log templates and word embeddings during offline training. On the one hand, we apply unsupervised template extraction methods [5] to get templates from historical logs. LogIE later uses these templates in the online stage for matching and processing logs purposes. On the other hand, to rank arbitrary summaries, LogSummary applies Log2Vec [32], a commonly used technique to generate log embeddings in the online system, to learn the domain-specific semantics of offline logs and generates a new embedding for unseen (OOV) words at runtime. Afterward, these trained embeddings are adopted to rank summaries online.

In the online stage, LogSummary updates new templates using online log parsing methods [4]. Then, it is followed by two consecutive steps, LogIE and ranking summaries. The LogIE (described in Section V-A) is a mechanism to generate triples from real-time logs and historical log templates. It solves the challenge that logs contain domain-specific text and outputs primary summaries of logs. In general, NLP-based summarization methods can be divided into abstractive and extractive methods [33]. The abstractive methods require supervised data, while the extractive methods are usually unsupervised. Since it is difficult to obtain a labeled log dataset, we use an extractive method to extract semantic triples in LogIE. Additionally, existing NLP-based summarization methods are typically designed for normal sentences, and logs are usually different from normal sentences, making it ineffective to use them directly to obtain log summarization. So we use domain knowledge to pre-process the log template before generating the summarization in LogIE. Note that applying NLP-based methods for log summarization is not the main contribution of our work. Moreover, LogIE saves mapping caches between triples and templates, which speeds up log processing and solves the processing speed challenge imposed by the massive amount of logs. Eventually, LogSummary ranks triples by adopting TextRank, which meets the requirements of operators reading critical summaries first rather than following the program execution logs order.

V. Algorithm

A. LogIE

In order to accurately and efficiently extract valuable information from logs, we propose LogIE (Log Information Extraction). LogIE performs open information extraction on logs, extracting triples relating entities and arguments through relation or predicate. To achieve this, it combines both Rule Extraction (RE) and OpenIE to extract the triples. To make the process fast and efficient, LogIE adopts templates to improve and speed up the information extraction of logs. Note that templates are automatically extracted by existing approaches [4], [5]. LogIE learns triples from the log templates, so template matching can be used to produce the LogIE triples output. LogIE is a framework composed of four main components, and we describe and explain how they work using the simple example in Fig. 6.

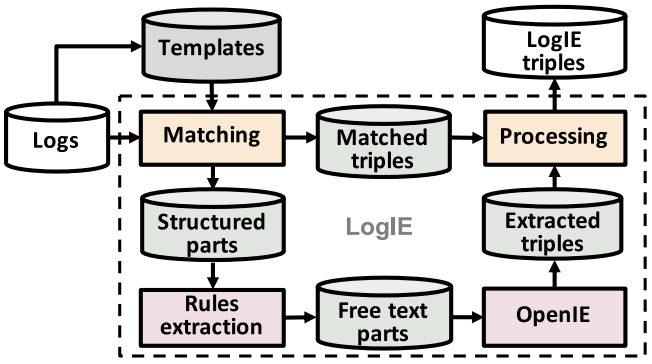

Fig. 4. Detailed workflow of the LogIE in LogSummary (Fig. 3).

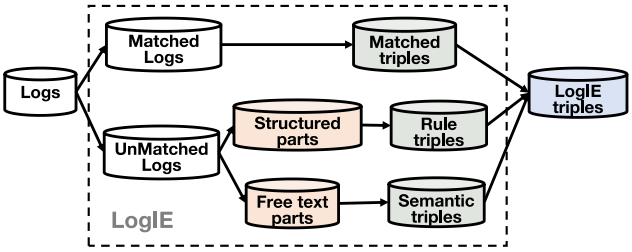

Fig. 5. Objectflow of the LogIE in LogSummary (Fig. 3).

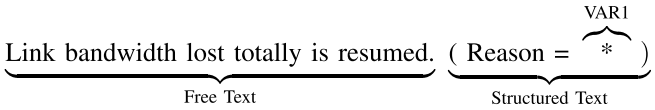

Fig. 6. Log template example.

1) Matching & Processing: Matching and Processing are the overarching components of LogIE supported by the RE and the OpenIE components that perform the triple extraction. The detailed workflow and object flow of the LogIE are shown in Fig. 4 and Fig. 5, respectively. LogIE takes both raw logs and templates as its input. It performs template matching on the input raw logs. Using Fig. 6 as an example, if a log is matched with this template, LogIE retrieves the previously extracted triples for this given log-type and substitutes the variables present in the triples by their actual value obtained from the raw log. These variables are usually identifiers, values, or addresses [1]. Therefore once a log is received, if a template is matched by the matching component, it will be processed directly by the processing component and output its LogIE triples. This way LogIE can effectively and efficiently yield OpenIE triples in an online manner. Since the goal of LogIE is to get structured information from logs, we treat all these cases equally by substituting them with a dummy token to be considered an entity or part of an argument, *e.g.,* "VARX", where X is the ordinal of the variable within the template. In the case that the log is not matched to any template, a new template needs to be extracted [4], [29]. Since LogIE is meant to be run online, it requires a template extraction and matching method that can be incrementally updated online. For this reason, we incorporate LogParse into the Matching component. The new template is then split into subparts as shown in Fig. 6, based

TABLE I
THREE RULES IN RE

Rule	Log	Rule Triple
"="	critical input interrupt (unit = 0x0b)	(unit, is, 0x0b)
":"	instruction address : 0x00004ed8	(address, is, 0x00004ed8)
"–"	mmcs_db_server has been started : ./mmcs_db_server –useDatabase BGL	(useDatabase, is, BGL)

on rules predefined accordingly to the source log. These subparts are then handled by the RE and OpenIE components to extract a new set of triples from it. The RE component would handle the structured parts, while the OpenIE one handles the free text parts. The output triples are then stored and passed to the Processing component to produce the final LogIE triples output.

2) RE for Rule Triples: The purpose of the RE component is to make the most out of the structure present in logs, namely the structured text part from the example in Fig. 6. According to our observation, there are some rules for systems to print logs. Therefore it becomes easier to define rules to extract part of the information present precisely. For example, in our implementation, we use three different rules, where all three cover different ways of representing entity-value pairs, as shown in Table I. For these cases, entity-value pairs are usually separated by an equals "=" or a colon ":" symbol. Another common case is formatting such information in the same way command line arguments are specified in a command line interface program. The RE component processes unstructured logs by first extracting triples from the non-free text parts of the logs before the OpenIE component processes their remaining free text parts. When non-free text is processed, OpenIE component will be more accurate and efficient in processing the remaining free text. Besides, the output triples of the LogIE's RE component could be used to provide further details of the log stream in a readable structured manner or store structured information (entity-value pairs) for further data mining. For example, when operators see a summarization (Interface, changed to, down) in a visual interface, they usually want to know which interface has changed, and rule triples can automatically provide this information if it is presented in a structured way in the log. Note that rule triples are not used for our ranking algorithm because they contain only simple "is" relation instead of rich semantic information.

3) OpenIE for Semantic Triples: Operators pay attention to "entities", "events", and the "relation" between them when they read logs which make these the most important pieces of information to be considered for log summarization. OpenIE [16] is usually used to extract relational triples, which is exactly what the operators need since they are both structured in a human-readable way [17], and a reduced version of the original logs. After rule triples were extracted using templates, the remainder free text is passed on to the OpenIE component. There has been substantial progress in OpenIE approaches since it was proposed by Banko et al. [16]. These methods take free text as input and yield OpenIE triples as the output, formed by two arguments related by a predicate, *e.g.,* ("Link bandwidth", "is", "resumed"). OpenIE methods achieve their objective by leveraging the underlying semantic structure of the sentences for a given language, enabling them to find the arguments present and the predicates that relate to them. Therefore, We leverage existing OpenIE approaches in our implementation to fulfill the OpenIE component requirement of the LogIE framework. Since LogIE is a framework, none of its components, including OpenIE, are tied to any implementation in particular. Besides, many short logs do not have the whole three elements of triples, *e.g.,* do not contain an entity, OpenIE can also generate "triples" with less than three elements. The outputs of the OpenIE component are the semantic triples, LogSummary will rank them and generate the summary later. As you will see in our evaluation in Section VI, we incorporate the main OpenIE methods from the literature into our work as both baselines for LogIE and as part of the LogIE framework for evaluation.

B. Ranking Summaries

As aforementioned, logs, which record software's status in real-time, usually suffer from redundancy and repetition. Traditionally, operators need to read raw logs and extract valuable information manually. However, it's labor-intensive and time-consuming. The goal of the LogSummary is to help operators to read/understand logs faster. After the LogIE stage, we obtain triples, the minimum units of semantic information within logs. In this section, we introduce a mechanism to rank the triples based on their informativeness. Operators generally hope to find out the importance of each triple by measuring the connection between each triple and other triples. For a set of logs, most algorithms ignore the semantics and other elements of its words, and simply treat a triple as a collection of words. And each word appears independently and does not depend on the other. However, in the log analysis domain, different word combinations have different meanings. Operators usually use knowledge drawn from entire logs to make local ranking decisions. Therefore, we integrate the information from the global corpus into the sorting algorithm in the form of word vectors, and through the combination of the sorting algorithm and word vectors, iteratively scores sentences and sorts them according to the score.

1) Triple Representation: Firstly, we propose a method to represent triples with domain-specific semantics. Log2Vec [32] enables generalization to domain-specific words, which is achieved by integrating the embedding of lexical and relation features into a low-dimensional Euclidean space. By training a model over the existing vocabulary, Log2Vec [32] can later use that model to predict the embedding of any words, even previously unseen words at runtime. Therefore, we apply the technique in Log2Vec to represent triples generated from logs. Leveraging its previous components, we convert any word in the triples into a word embedding vector and generate the triple's vector, which is the arithmetic mean of its word vectors.

Algorithm 1 Ranking Algorithm

Require: A semantic information triple set *ST*, the number of triple candidates *k* and word embedding set *WE*
Ensure: Ranked summaries *S*

```
Create a triple vector set TV
for each triples st of ST do
    Create a temporary empty triple vector tv
    Let a integer variable len record the number of words in st
    for each word w of st do
        Find the corresponding word vector wv for w in WE
        Plus the current word vector wv to the temporary triple vector tv
    end for
    Get the average vector av by dividing tv by len and regard av as the triple vector st
    Append the triple vector av to TV
end for
Init a matrix of transition probability M by calculating cosine similarity between all tv pairs in TV
Convert M to a weighted graph G = (V, E).
Get the triple scores TS by applying Formula (1) to G
Sort the triples in reverse order by scores in TS, and the top k triples as the final summaries S.
return S
```

2) Ranking Triples: We propose a method to rank log summaries in this section. Its workflow is shown in Algorithm 1. Firstly, we build a graph associated with the logs, where the graph vertices are representative of the units to be ranked. For the application of triple extraction, the goal is to rank entire semantic triples, and therefore a vertex is added to the graph for each triple in logs. Same as sentence extraction, we define a relation that determines a connection between two triples if there is a "similarity" relation between them, where the "similarity" is measured as the cosine similarity [34] of two triples. Note that other similarity measures (*e.g.,* Euclidean distance [35]) are also possible. Such a relation between two triples can be seen as a process of "recommendation". Given a triple that addresses certain concepts in logs, it is "recommended" to refer to other triples in the logs that address similar concepts. Therefore, a similarity link is drawn between any two triples.

Unlike the unweighted graphs in PageRank [36], we need to build weighted graphs. The resulting graph is highly connected, with a weight associated with each edge, indicating the strength of connections established between various triple pairs in logs. Therefore, the logs are represented as a weighted graph (Fig. 7 shows a weighted graph for the case study in Section VI-C). Formally, let $G = (V, E)$ be a directed graph with the set of vertices V and a set of edges E, where E is a subset of $V * V$. For a given vertex V_i, let $In(V_i)$ be the set of vertices that point to it, and let $Out(V_i)$ be the set of vertices that vertex V_i points to. Then, we adopt the formula in TextRank [37], which is for graph-based ranking that takes into account edge weights when computing the score associated with a vertex in the graph. Textrank's formula is defined to integrate vertex weights.

$$WS(V_i) = (1-d) + d * \sum_{V_j \in In_{V_i}} \frac{w_{ji}}{\sum_{V_k \in Out_{V_j}} w_{jk}} WS(V_j) \quad (1)$$

where d is a damping factor that can be set between 0 and 1.

After the triple-based TextRank [37] is run on the graph, semantic triples are sorted in reverse order of their score.

Note that although the summaries of LogSummary are highly compressed, it has a different goal from other log compression applications (*e.g.,* LogZip [3]). Other log compression applications aim to store logs, while our summaries are more readable for operators.

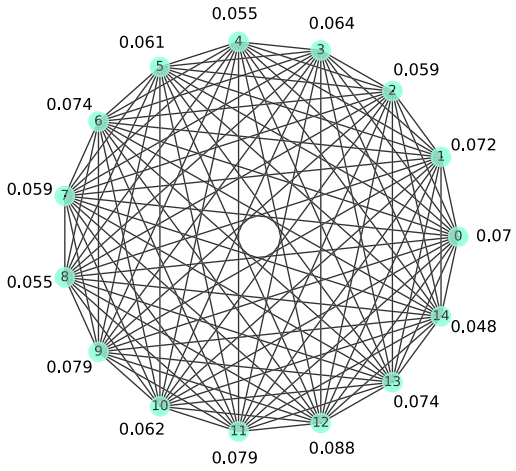

Fig. 7. Weighted graph of case study on real-world switch logs.

VI. Experiments

In this section, we report all experiments conducted to evaluate the effectiveness of the proposed LogSummary. We evaluate LogSummary from two aspects. Firstly, we evaluate the accuracy of LogIE on information extraction and compare it with common information extraction approaches. Next, we evaluate the accuracy of ranked summaries on the gold references and compare the result with the baselines.

A. Experimental Setting

1) Datasets: We conduct experiments over four public log datasets from several services: BGL logs [38], HDFS logs [7], HPC logs [39], and Proxifier logs [3].

Since the lack of a publicly-available summary gold standard hinders the automatic evaluation, we manually label the above public log datasets and make them available. In this paper, we provide two kinds of gold standard datasets. For information extraction, we label all templates for all logs ([5] only label templates for 2000 logs per dataset) and label OpenIE triples leveraging semantic information and domain knowledge. In detail, four researchers manually extracted the triples from the templates and gave each template a variable number of semantic triples based on experience. Then, they discussed the annotation results and revised them again

TABLE II
EXAMPLES OF MANUAL LABELING

Log Template	Triples
- * error : Could not connect through proxy proxy.cse.cuhk.edu.hk : 5070 - Proxy server cannot establish a connection with the target , status code *	(“”, “Could not connect through”, “proxy”), (“Proxy server”, “cannot establish”, “a connection”)
Connections do not originate from the applications. A third party network filter (e.g. antivirus) is detected	(“Connections”, “do not originate from”, “the applications”), (“A third party network filter”, “is detected”, “”)

TABLE III
DETAIL OF THE SERVICE LOG DATASETS

Datasets	Description	# of logs
BGL	Blue Gene/L supercomputer log	4,747,963
HDFS	Hadoop distributed file system log	11,175,629
HPC	High performance cluster log	433,489
Proxifier	Proxifier software log	21,329

TABLE IV
RESULTING NUMBER OF MANUALLY IMPROVED TEMPLATES AND MANUALLY EXTRACTED OPENIE TRIPLES FROM THE SOURCE DATASETS FOR BUILDING THE OPENIE GOLD STANDARD DATASET

Source	Templates	Triples
BGL	263	831
HDFS	50	87
HPC	85	199
Proxifier	14	36
Total	412	1,153

to ensure consistency. The examples of manual labeling are listed in Table II. To evaluate log summarization, we choose 100 groups of 20 contiguous logs per dataset and generated their summaries manually.

The detailed information on the above datasets is listed in Table III and Table IV.

2) Experimental Setup: We conduct all experiments on a Linux server with Intel Xeon 2.40 GHz CPU. We implement LogSummary with Python 3.6 and make it open-source.

B. Evaluation on LogIE

We intrinsically evaluate the LogIE framework to choose its best implementation and compare it to the baselines. We evaluate the main OpenIE methods from the literature as baselines and incorporate them as the OpenIE component of LogIE. For this task, we build and open-source a dataset of log information extraction triples based on public logs. We also use this gold standard dataset to evaluate an improved version of LogIE which uses manually improved templates instead of an online template extraction approach. This serves as an ablation test showing the influence of the template quality on LogIE's performance.

1) Triples Gold Standard Dataset: For this dataset, we employ it as the gold standard for building and evaluating the different approaches within LogIE. The process of building it can be divided into three parts: obtaining the logs data from different services, extracting and improving templates used to assist the triples extraction, and manual annotation.

We source the logs from different types of systems and services. Four described in Table VII are open-source and one comes from real-world switch logs.

As part of building the gold propositions dataset, we extract templates from all source logs using LogParse [4]. Then, we manually extract OpenIE triples from the logs. For each log, we manually extract relational triples of the form (arg_1, r, arg_2), meaning that arg_1 is related to arg_2 by predicate r. We aim to keep the form (subject, predicate, object) in this order. For this purpose, we considered both the domain knowledge and the semantic structure. We make several considerations: We extract (subject, predicate, object) triples in this order. Where applicable, we make prepositions part of the predicate. It's required to have at least the predicate and the subject or object present to extract a triple. We make linking verbs the predicate of a triple where applicable. Conjunctions, such as “and” or “or,” are split into several triples or combined into a single one where the conjunction is part of the predicate. For the cases of apposition, we define an “is” relation that would serve as an “is-a” relation, which is usually used in ontology building.

Lastly, we leverage domain knowledge to extract the values of arguments or attributes as well as the instances of different entities. Usually, these would show up as an “=” or a “:” in the logs. In these cases, we also considered an “is” relation to represent the relation between the two. Additionally, arguments are also represented in the format in which command-line arguments are written. In these cases, we also use an “is” relation and create a second argument “set” for flags.

2) Task Formulation: As explained in Section III, OpenIE intends to obtain all relations present in a given sentence or corpus together with the arguments or entities related by such relations in a structured manner. Likewise, the goal of LogIE is to extract the relations present within each log, which are used as the minimum unit of information from each log. Specifically, given a stream of raw logs as the input, relational triples of the form $(\text{arg}_1, \text{relation}, \text{arg}_2)$ are to be extracted for each relation present within each log. This task will be tested against the gold standard we propose in Section VI-B1 and evaluated as detailed in the following Section VI-B3.

3) Metrics and Baselines: The main challenge to intrinsically evaluate LogIE, similarly to cases that are common in NLP, is that we need to allow different OpenIE triples extractions to be considered acceptable for the same gold proposition. For this reason, we follow a similar approach to that of Stanovsky and Dagan [40] in their OpenIE benchmark. Inspired by He et al. [41], where the syntactic heads

TABLE V
TEST ACCURACY ON THE LOG OPENIE TRIPLES GOLD REFERENCE DATASET OF PUBLIC LOGS

Logs	Method	Precision	Recall	F1
BGL	**LogIE**	**0.918**	**0.864**	**0.89**
	OpenIE5	0.788	0.733	0.760
	Stanford	0.685	0.753	0.717
	Ollie	0.552	0.633	0.590
	PredPatt	0.463	0.638	0.536
	ClausIE	0.447	0.602	0.513
	PropS	0.000	0.000	0.000
HDFS	**LogIE**	**0.98**	**0.459**	**0.626**
	OpenIE5	0.271	0.220	0.243
	Stanford	0.184	0.210	0.196
	Ollie	0.003	0.079	0.006
	PredPatt	0.171	0.177	0.174
	ClausIE	0.159	0.530	0.244
	PropS	0.000	0.000	0.000
HPC	**LogIE**	**0.859**	**0.667**	**0.751**
	OpenIE5	0.567	0.123	0.202
	Stanford	0.691	0.349	0.464
	Ollie	0.290	0.285	0.287
	PredPatt	0.591	0.556	0.573
	ClausIE	0.588	0.648	0.616
	PropS	0.000	0.000	0.000
Proxifier	**LogIE**	**0.869**	**0.812**	**0.839**
	OpenIE5	0.759	0.204	0.322
	Stanford	0.831	0.254	0.389
	Ollie	0.556	0.194	0.288
	PredPatt	0.061	0.106	0.078
	ClausIE	0.247	0.719	0.368
	PropS	0.000	0.000	0.000

of the predicate and the arguments from a given extraction should match those of the corresponding gold proposition, they define a more lenient approach that considers their token-level overlap instead. Therefore, we use an approach similar to theirs[3] to calculate the precision and recall of the evaluation. Among the main differences, we do not propagate a match to all matching predicates but thoroughly test all triples against all gold propositions instead. We don't produce a confidence score for each triple, so we don't calculate AUC scores. The main metric we consider for comparison between the approaches is the F1 score. The metrics are calculated as follows: $precision = \frac{\#\ correct\ extractions}{\#\ extractions}$, $recall = \frac{\#\ recalled\ gold\ propositions}{\#\ gold\ propositions}$, $F1 = \frac{2 \times precision \times recall}{precision + recall}$.

We compare LogIE with six Open Information Extraction methods, namely, ClausIE [42], Ollie [43], OpenIE5,[4] PredPatt [44], PropS [44], and Stanford OpenIE [45].

4) Experimental Results: We evaluate LogIE and compare it against its manually augmented version and the six OpenIE baselines in Table V on the four public logs described in Table III. LogIE learns online templates generated by LogParse [4]. However, to perform an ablation test, we also compare LogIE improved, an augmented version using manually improved templates that were produced as part of the gold standard dataset introduced in Section VI-B1. Then each baseline is plain OpenIE approaches used directly on the raw logs. LogIE consistently produces better results across all public logs when compared to the baselines. This is because LogIE's pipeline approach is optimized to take advantage of both the structure and the free text present in logs. On the other hand, plain OpenIE methods are meant to be used directly on free natural language text. As you will see in Table V, even though both versions of LogIE are consistently superior to the baselines, there are cases where their results could be comparable, such as on BGL or HPC. The more free text is present in the logs, the easier it is for plain OpenIE methods to generate correct OpenIE triples. However, as we show in Table VI, applying plain OpenIE methods on the logs is inefficient compared to LogIE, which leverages the templates used as input, and the high speed of template matching using tries to produce the OpenIE triples output from the raw logs. The throughput of LogIE is over 200X that of applying plain OpenIE, and this benefits mainly from template matching. Nonetheless, the performance of LogIE is sensitive to the accuracy of the templates used as input, as shown in the ablation test comparison. As demonstrated by the performance of LogIE Improved, the more accurate templates are, the better the performance of LogIE. Further, LogIE leverages either the structure of the log or the semantic information of the unstructured text within logs to extract information. If there is no rich information in the structure or details within the log are omitted to make it shorter, its performance is also affected. This is the case for the HDFS logs, where the structured parts don't provide rich information and the natural language implicitly refers to the arguments, which is not picked up by LogIE. This affects its results with a low recall as seen in Table V. In turn, this affects the output of LogSummary, given the pipeline nature of the framework.

TABLE VI
COMPARISON OF SPEED MEASURED IN LOGS PER SECOND BETWEEN LOGIE AND THE PLAIN OPENIE METHODS WHEN PROCESSING THE INPUT LOGS MEASURED OVER THIRTY RUNS FOR EACH OPENIE METHOD AND EACH LOGS DATASET

Approach	**Throughput (logs / s)**	
	mean	std
LogIE	8,550.66	1,909.62
OpenIE Methods	39.05	36.19

C. Evaluation on Ranking Summaries

1) Metrics and Baselines: To automatically evaluate the log summarization performance of different approaches, we use ROUGE [46]. The ROUGE metric measures the summary quality by counting the overlapping units between

[3]https://github.com/gabrielStanovsky/oie-benchmark
[4]https://github.com/dair-iitd/OpenIE-standalone

TABLE VII
LOG SUMMARIZATION PERFORMANCE AND COMPRESSION RATIO (CR) OF LOGSUMMARY COMPARED TO ITS BASELINES

Logs	Method	Precision	Recall	F1	CR
BGL	**LogSummary**	**0.815**	**0.703**	**0.725**	**0.026**
	CD-LDA	0.382	0.076	0.119	0.130
	TextRank	0.893	0.238	0.347	0.144
	TF-IDF	0.383	0.354	0.332	0.024
HDFS	**LogSummary**	**0.759**	**0.432**	**0.538**	**0.015**
	CD-LDA	0.220	0.045	0.074	0.225
	TextRank	0.602	0.079	0.135	0.230
	TF-IDF	0.193	0.176	0.179	0.033
HPC	**LogSummary**	**0.819**	**0.911**	**0.840**	**0.037**
	CD-LDA	0.530	0.110	0.175	0.251
	TextRank	0.904	0.265	0.365	0.208
	TF-IDF	0.487	0.506	0.472	0.039
Proxifier	**LogSummary**	**0.879**	**0.857**	**0.864**	**0.045**
	CD-LDA	0.332	0.088	0.135	0.099
	TextRank	0.663	0.050	0.093	0.275
	TF-IDF	0.281	0.324	0.290	0.023

the generated summary and reference summaries. In our scenario, different operators may manually generate summaries in different orders of words/phrases. Therefore, we apply ROUGE-1 to evaluate performance. Following the common practice [47], we report the precision, recall, and F1 score for ROUGE-1, where $precision = \frac{\#\ overlapping\ words}{\#\ words\ in\ goldreference}$, $recall = \frac{\#\ overlapping\ words}{\#\ words\ in\ automatic\ summary}$ and $F1socre = \frac{2\times precision\times recall}{precision+recall}$ We obtain the metrics using open-source package.[5] We apply the compression ratio, *i.e.,* $\frac{size\ of\ summaries}{size\ of\ original\ logs}$, to evaluate the log compression performance. We compare LogSummary with three baseline extractive summarization methods, namely, TF-IDF [48], LDA [15], and TextRank (sentence summary) [37]. We implement TF-IDF and LDA with scikit-learn, a Python library for software machine learning, and implement TextRank with NetworkX, a Python library for studying graphs and networks. Specifically, for the TF-IDF, we consider the log sequence as a corpus and each log as a document and then calculate the TF-IDF weight. We select the k words with the highest weight from a log as the summarization of the log sequence. For LDA, we set the number of topics to 10 and select the k words with the highest weight for each topic as the summarization of the topic, and the summarization of the log sequence is the set of all topic summarizations. For TextRank, we use Log2Vec [32] to generate a vector representation of each log and then rank the importance of the logs based on the TextRank algorithm. We select the most important k logs as the summarization of the log sequence. We set $k = 10$ in the experiments.

2) Experimental Results: We compare LogSummary with three baselines on four public datasets. For LogSummary, we choose the top-5 semantic triples from online logs. Table VII shows the comparison results of LogSummary and three baselines. Overall, LogSummary achieves the best summarization accuracy among the four methods. Both TF-IDF and LDA, however, have low F1 scores (< 0.5) on all four datasets because TF-IDF and LDA generate summaries by extracting keywords, which dismisses valuable information in raw logs. Although TextRank achieves relatively high precision (*e.g.,* 0.904 on the BGL dataset), the high precision is at the cost of low recalls. For instance, on the Proxifier dataset, the recall of TextRank is only 0.050. Because there are many similar logs with different variables, when employed on their own, TextRank may choose many logs of the same type and ignore other types of logs. On the contrary, LogSummary uses LogIE to extract triples from logs as an intermediate representation, which is more fine-grained than each complete log, before applying TextRank achieving a ≈4.6 times higher recall.

Table VII evaluates the compression ratio for log summarization on four datasets. We find that LogSummary achieves an average compression ratio of 3.1%, which will vastly reduce the reading and understanding load of operators.

The results mean that the outputs of LogSummary are not only readable but also highly compressed.

[5]https://github.com/pltrdy/rouge

D. Threats to Validility

The LogSummary framework leverages each of its components to produce accurate summaries automatically. However, its pipeline nature makes each component depends on the quality of the output from the previous ones. Sometimes, we should improve the templates manually since template extraction is not perfectly precise. These imprecisions meant there would be redundant templates extracted from the logs. Additionally, the variables may not have been detected properly in some cases. Therefore, the quality of the templates may affect the triples extracted by LogIE, which in turn affects the representations built by Log2Vec [32] which are used by TextRank [37] to produce the ranked summaries. Nonetheless, each of the components provides significant benefits over their baselines. LogIE produces triples at over 200 times the throughput of plain OpenIE methods, which serve as the intermediate result that LogSummary leverages to achieve ≈4.6 times the recall of TextRank [37], which is the best-performing baseline.

Further, we consider four open source log datasets as part of the evaluation from software systems. Our approaches outperform their baselines in all of them, which shows the generalizability of LogSummary. However, it may encounter challenges when dealing with short logs with non-conforming syntax or complex application-layer logs with lots of parameters in their free text part. For example, operators need to create complex rules for the Rule Extraction part when faced with complex logs with many parameters. When a target system has a large number of grammatically completed logs, LogSummary will play an important role in it.

VII. DISCUSSION

To further evaluate the performance and demonstrate the robustness of LogSummary, we do a case study on real-world logs, which are generated by switches deployed in a top cloud service provider. We select and label one million switch logs in the same manner we do for the public datasets as described

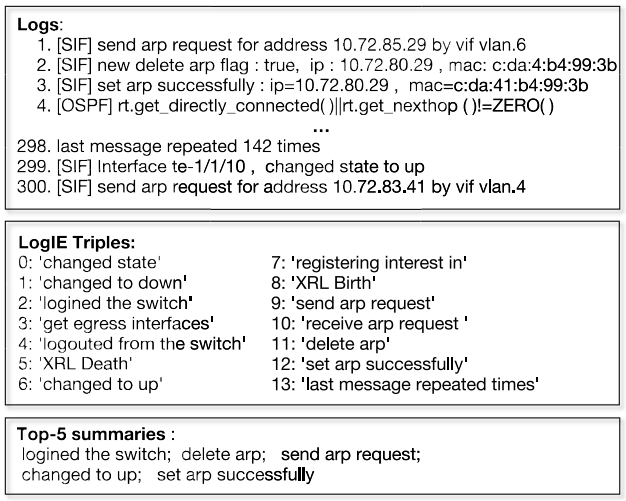

Fig. 8. A case study of LogSummary on switch logs.

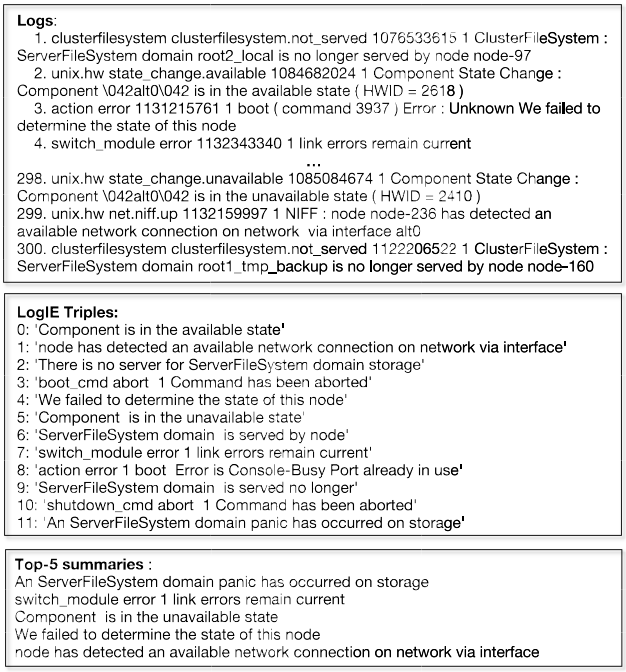

Fig. 9. A case study of LogSummary on HPC logs.

in Section VI-B1. Here results are consistent with the experiments on the public log datasets. Likewise, given that the proportion of the free text is higher within this log dataset, a complex OpenIE approach can also generate comparably accurate triples. Nonetheless, both LogIE approaches outperform all baselines at a throughput over 200 times higher. For this real-world dataset, LogIE achieves an F1 of 0.831, whereas other OpenIE methods get an average F1 of 0.414. At the same time, LogSummary helps operators to increase their productivity by orders of magnitude compared to the existing combination of manual and rule-based methods.

We conduct a case study of LogSummary on real-world switch logs to visualize the intermediate steps and showcase its results. As shown in Fig. 8, we randomly select 300 logs from real-world switch logs, apply LogIE to extract triples, and generate summarization with LogSummary. The corresponding weighted triple graph from TextRank is shown in Fig. 7. Besides, we also conduct a case study on HPC to ensure consistency with the evaluation. We randomly select 300 logs from the HPC dataset and apply the same processing flow as in switch logs. The result is shown in Fig. 9. Both summarization results are confirmed by operators, proving that LogSummary is useful.

There is abundant room for further progress in improving the ranking algorithm. We do not consider the order of logs in the ranking algorithm, which may have an impact on the log summarization. In further work, we can add order information to the ranking algorithm. For example, when calculating the weights between triples, we could inverse the weights to the average distance between them. We can also calculate the number of triples in a certain interval and update the weights proportional to it.

LogSummary can serve further downstream purposes, which we consider for future work. The triples of LogSummary could aid in the creation of knowledge graphs applied to perform automatic root cause analysis. Additionally, they could serve as an intermediate representation before other log analysis tasks. Suppose we add domain knowledge of anomaly logs to the triple ranking algorithm or use labeled triples to train a deep learning model that can indicate whether a triple is anomalous. In that case, LogSummary could produce a summary of anomalous logs for troubleshooting.

VIII. Conclusion

Logs play an important role in network and service maintenance. Operators still have to conduct log summarization for a suspicious log sequence in a manual or rule-based manner before taking action, even though many methods have been proposed for automatic log detection/diagnosis/prediction. In this paper, we propose LogSummary, a simple baseline framework for automatic summarization for large-scale online services. LogSummary combines LogIE, which accurately and efficiently obtains information extraction triples from logs, and a simple yet effective triple-ranking method utilizing the global knowledge learned from all historical logs. We perform extensive evaluation experiments to demonstrate LogSummary's performance in summarizing logs. Moreover, we have open-sourced LogSummary and the manually labeled gold standard references, hoping that they can benefit future research works. In this work, we take the first step towards automated log summarization in an interpretable and readable manner for online services and believe that LogSummary will benefit researchers for their future work in log summarization.

References

[1] A. Pi, W. Chen, S. Wang, and X. Zhou, "Semantic-aware workflow construction and analysis for distributed data analytics systems," in *Proc. 28th Int. Symp. High-Perform. Parallel Distrib. Comput.*, 2019, pp. 255–266.

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

[3] J. Liu, J. Zhu, S. He, P. He, Z. Zheng, and M. R. Lyu, "Logzip: Extracting hidden structures via iterative clustering for log compression," in *Proc. 34th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2019, pp. 863–873.

[4] W. Meng et al., "Logparse: Making log parsing adaptive through word classification," in *Proc. Int. Conf. Comput. Commun. Netw. (ICCCN)*, 2020, pp. 1–9.

[5] J. Zhu et al., "Tools and benchmarks for automated log parsing," in *Proc. 41st Int. Conf. Softw. Eng. (ICSE)*, 2019, pp. 121–130.

[6] Y. Liu et al., "Uniparser: A unified log parser for heterogeneous log data," in *Proc. ACM Web Conf.*, 2022, pp. 1893–1901.

[7] M. Du, F. Li, G. Zheng, and V. Srikumar, "DeepLog: Anomaly detection and diagnosis from system logs through deep learning," in *Proc. ACM SIGSAC Conf. Comput. Commun. Security (CCS)*, 2017, pp. 1285–1298.

[8] W. Meng et al., "Loganomaly: Unsupervised detection of sequential and quantitative anomalies in unstructured logs," in *Proc. 28th Int. Joint Conf. Artif. Intell.*, vol. 7, 2019, pp. 4739–4745.

[9] S. Zhang et al. "PreFix: Switch failure prediction in datacenter networks," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 2, no. 1, p. 2, 2018.

[10] X. Zhou et al., "Latent error prediction and fault localization for microservice applications by learning from system trace logs," in *Proc. 27th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2019, pp. 683–694.

[11] 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 *Proc. IEEE 32nd Int. Symp. Softw. Rel. Eng. (ISSRE)*, 2021, pp. 80–90.

[12] P. Dogga, K. Narasimhan, A. Sivaraman, and R. Netravali, "A system-wide debugging assistant powered by natural language processing," in *Proc. ACM Symp. Cloud Comput.*, 2019, pp. 171–177.

[13] E. Gentili, A. Milani, and V. Poggioni, "Data summarization model for user action log files," in *Proc. Int. Conf. Comput. Sci. Appl.*, 2012, pp. 539–549.

[14] J.-G. Lou, Q. Fu, S. Yang, J. Li, and B. Wu, "Mining program workflow from interleaved traces," in *Proc. 16th ACM SIGKDD Int. Conf. Knowl. Disc. Data Min.*, 2010, pp. 613–622.

[15] S. Satpathi, S. Deb, R. Srikant, and H. Yan, "Learning latent events from network message logs," *IEEE/ACM Trans. Netw.*, vol. 27, no. 4, pp. 1728–1741, Aug. 2019.

[16] M. Banko, M. J. Cafarella, S. Soderland, M. A. Broadhead, and O. Etzioni, "Open information extraction from the Web," in *Proc. CACM*, 2008, pp. 1–7.

[17] M. Mausam, "Open information extraction systems and downstream applications," in *Proc. IJCAI*, 2016, pp. 4074–4077.

[18] J. Qiang, Z. Qian, Y. Li, Y. Yuan, and X. Wu, "Short text topic modeling techniques, applications, and performance: A survey," *IEEE Trans. Knowl. Data Eng.*, vol. 34, no. 3, pp. 1427–1445, Mar. 2022.

[19] D. Gunter, B. L. Tierney, A. Brown, M. Swany, J. Bresnahan, and J. M. Schopf, "Log summarization and anomaly detection for troubleshooting distributed systems," in *Proc. 8th IEEE/ACM Int. Conf. Grid Comput.*, 2007, pp. 226–234.

[20] W. Kryściński, N. S. Keskar, B. McCann, C. Xiong, and R. Socher, "Neural text summarization: A critical evaluation," 2019, *arXiv:1908.08960*.

[21] W. Shang, M. Nagappan, A. E. Hassan, and Z. M. Jiang, "Understanding log lines using development knowledge," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2014, pp. 21–30.

[22] S. Kabinna, W. Shang, C.-P. Bezemer, and A. E. Hassan, "Examining the stability of logging statements," *Empir. Softw. Eng.*, vol. 23, no. 1, pp. 290–333, 2018.

[23] Y. Huo, Y. Su, B. Li, and M. R. Lyu, "SemParser: A semantic parser for log analysis," 2021, *arXiv:2112.12636*.

[24] S. He, Q. Lin, J.-G. Lou, H. Zhang, M. R. Lyu, and D. Zhang, "Identifying impactful service system problems via log analysis," in *Proc. 26th ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2018, pp. 60–70.

[25] V.-H. Le and H. Zhang, "Log-based anomaly detection without log parsing," in *Proc. 36th IEEE/ACM Int. Conf. Autom. Softw. Eng. (ASE)*, 2021, pp. 492–504.

[26] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet allocation," *J. Mach. Learn. Res.*, vol. 3, pp. 993–1022, Jan. 2003.

[27] R. Chen, Q. Gao, W. Ji, F. Long, and Q. Ling, "Network log analysis based on the topic word mover's distance," in *Proc. Chin. Control Decis. Conf. (CCDC)*, 2018, pp. 4082–4086.

[28] K. Otomo, S. Kobayashi, K. Fukuda, and H. Esaki, "Latent semantics approach for network log analysis: Modeling and its application," in *Proc. IFIP/IEEE Int. Symp. Integr. Netw. Manage. (IM)*, 2021, pp. 215–223.

[29] P. He, J. Zhu, Z. Zheng, and M. R. Lyu, "Drain: An online log parsing approach with fixed depth tree," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, 2017, pp. 33–40.

[30] S. Zhang et al., "Syslog processing for switch failure diagnosis and prediction in datacenter networks," in *Proc. IEEE/ACM 25th Int. Symp. Qual. Service (IWQoS)*, 2017, pp. 1–10.

[31] T. Mikolov, Q. V. Le, and I. Sutskever, "Exploiting similarities among languages for machine translation," 2013, *arXiv:1309.4168*.

[32] W. Meng et al., "A semantic-aware representation framework for online log analysis," in *Proc. IEEE Int. Conf. Comput. Commun. (ICCCN)*, 2020, pp. 1–7.

[33] M. Gambhir and V. Gupta, "Recent automatic text summarization techniques: A survey," *Artif. Intell. Rev.*, vol. 47, no. 1, pp. 1–66, 2017.

[34] L. Gravano, P. G. Ipeirotis, N. Koudas, and D. Srivastava, "Text joins for data cleansing and integration in an RDBMS," in *Proc. 19th Int. Conf. Data Eng.*, 2003, pp. 729–731.

[35] P.-E. Danielsson, "Euclidean distance mapping," *Comput. Graph. Image Process.*, vol. 14, no. 3, pp. 227–248, 1980.

[36] L. Page, S. Brin, R. Motwani, and T. Winograd, "The pagerank citation ranking: Bringing order to the Web," Stanford InfoLab, Stanford, CA, USA, Rep., 1999.

[37] R. Mihalcea and P. Tarau, "TextRank: Bringing order into text," in *Proc. Conf. Empir. Methods Nat. Lang. Process.*, 2004, pp. 404–411.

[38] W. Xu, L. Huang, A. Fox, D. Patterson, and M. I. Jordan, "Detecting large-scale system problems by mining console logs," in *Proc. ACM SIGOPS Symp. Oper. Syst. Principles*, 2009, pp. 117–132.

[39] Q. Lin, H. Zhang, J.-G. Lou, Y. Zhang, and X. Chen, "Log clustering based problem identification for online service systems," in *Proc. 38th Int. Conf. Softw. Eng. Compan. (ICSE)*, 2016, pp. 102–111.

[40] G. Stanovsky and I. Dagan, "Creating a large benchmark for open information extraction," in *Proc. EMNLP*, 2016, pp. 2300–2305.

[41] L. He, M. Lewis, and L. Zettlemoyer, "Question-answer driven semantic role labeling: Using natural language to annotate natural language," in *Proc. EMNLP*, 2015, pp. 643–653.

[42] L. Del Corro and R. Gemulla, "ClausIE: Clause-based open information extraction," in *Proc. WWW*, 2013, pp. 1–11.

[43] Mausam, M. Schmitz, S. Soderland, R. Bart, and O. Etzioni, "Open language learning for information extraction," in *Proc. EMNLP-CoNLL*, 2012, pp. 523–534.

[44] G. Stanovsky, J. Ficler, I. Dagan, and Y. Goldberg, "Getting more out of syntax with PropS," 2016, *arXiv:1603.01648*.

[45] G. Angeli, M. J. J. Premkumar, and C. D. Manning, "Leveraging linguistic structure for open domain information extraction," in *Proc. ACL*, 2015, pp. 344–354.

[46] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries," in *Text Summarization Branches Out*. Barcelona, Spain: Assoc. Comput. Linguist., Jul. 2004, pp. 74–81.

[47] K. Ganesan, "Rouge 2.0: Updated and improved measures for evaluation of summarization tasks," 2018, *arXiv:1803.01937*.

[48] S. Lee and H.-J. Kim, "News keyword extraction for topic tracking," in *Proc. 4th Int. Conf. Netw. Comput. Adv. Inf. Manage.*, vol. 2, 2008, pp. 554–559.

Weibin Meng received the B.S. degree in software engineering from Jilin University, Changchun, China, in 2016, and the Ph.D. degree from the Department of Computer Science and Technology and the Institute for Network Sciences and Cyberspace, Tsinghua University, Beijing, China, in 2021. He is currently a Principal Engineer with Huawei. His research interests include anomaly detection, log analysis, and network security.

Federico Zaiter received the B.S. degree in software systems engineering from Universidad ORT Uruguay in 2017, and the M.S. degree in computer science from the Department of Computer Science and Technology, Tsinghua University, Beijing, China, in 2020. He is currently a Lead Machine Learning Engineer with Tryolabs. He is also a Lecturer with the Department of Artificial Intelligence and Big Data, Universidad ORT Uruguay. His research interests include log analysis for anomaly detection and troubleshooting within AIOps.

Yuzhe Zhang received the B.S. degree in software engineering from Nankai University in 2020, where he is currently pursuing the M.S. degree with the College of Software. His research interests include deep learning and anomaly detection.

Ying Liu (Member, IEEE) received the Ph.D. degree in applied mathematics from Xidian University in 2001. She is currently a Full Professor with the Institute for Network Sciences and Cyberspace, Tsinghua University. Her research interests include multicast routing, network architecture, and router design and implementation.

Shenglin Zhang (Member, IEEE) received the B.S. degree in network engineering from Xidian University, Xi'an, China, in 2012, and the Ph.D. degree 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 in data center networks.

Shimin Tao received the M.S. degree from the Beijing University of Aeronautics and Astronautics, Beijing, China. He is currently a Technology Expert with Huawei 2012 Laboratory. The main research direction is AIOps, machine translation, and natural language processing.

Yichen Zhu received the B.S. degree in statistic from University of Toronto in 2020. He is currently a Researcher with Midea Group. His research interests are broadly in efficient deep learning and automated machine learning.

Tao Han received the B.S. degree in 2002. He is currently the Principal Architect with Huawei Qiankun, Huawei. His research interests are autonomous driving networks and security.

Yongpeng Zhao received the B.S. degree in 2001. He is currently the Principal Architect with NCE Data Communication Domain, Huawei. His research interests are autonomous driving networks and cloud computing.

En Wang (Member, IEEE) received the Ph.D. degree in computer science and technology from Jilin University, Changchun, in 2016, where he is currently a Professor with the Department of Computer Science and Technology. He is also a Visiting Scholar with the Department of Computer and Information Sciences, Temple University, Philadelphia. His current research focuses on crowdsensing, data mining, and mobile computing.

Yuzhi Zhang received the B.S. and M.S. degrees 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 the 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 (Senior Member, IEEE) received the M.S. degree in computer science from the Department of Computer Science and Technology, Tsinghua University in 2000, and the Ph.D. in computer science from the Computer Science Department, University of California at Los Angeles, Los Angeles, in 2005. He is currently an Associate Professor with the Department of Computer Science and Technology, Tsinghua University. His research interests include network and service management in general. He is an ACM senior member.