Identifying Linked Incidents in Large-Scale Online Service Systems

Yujun Chen*
Microsoft Research
Beijing, China

Xian Yang
Hong Kong Baptist University
Hong Kong, China

Hang Dong
Microsoft Research
Beijing, China

Xiaoting He*
Chinese Academy of Sciences
Beijing, China

Hongyu Zhang
The University of Newcastle
NSW, Australia

Qingwei Lin†
Microsoft Research
Beijing, China

Junjie Chen
College of Intelligence and Computing, Tianjin University
Tianjin, China

Pu Zhao
Yu Kang†
Microsoft Research
Beijing, China

Feng Gao
Zhangwei Xu
Microsoft Azure
Redmond, USA

Dongmei Zhang
Microsoft Research
Beijing, China

ABSTRACT

In large-scale online service systems, incidents occur frequently due to a variety of causes, from updates of software and hardware to changes in operation environment. These incidents could significantly degrade system's availability and customers' satisfaction. Some incidents are linked because they are duplicate or inter-related. The linked incidents can greatly help on-call engineers find mitigation solutions and identify the root causes. In this work, we investigate the incidents and their links in a representative real-world incident management (IcM) system. Based on the identified indicators of linked incidents, we further propose **LiDAR** (Linked Incident identification with DAta-driven Representation), a deep learning based approach to incident linking. More specifically, we incorporate the textual description of incidents and structural information extracted from historical linked incidents to identify possible links among a large number of incidents. To show the effectiveness of our method, we apply our method to a real-world IcM system and find that our method outperforms other state-of-the-art methods.

CCS CONCEPTS

• **Software and its engineering** → **Maintaining software**; • **Computer systems organization** → **Cloud computing**.

KEYWORDS

Linked incidents, incident management, online service system

ACM Reference Format:
Yujun Chen, Xian Yang, Hang Dong, Xiaoting He, Hongyu Zhang, Qingwei Lin, Junjie Chen, Pu Zhao, Yu Kang, Feng Gao, Zhangwei Xu, and Dongmei Zhang. 2020. Identifying Linked Incidents in Large-Scale Online Service Systems. In *Proceedings of the 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering (ESEC/FSE '20), November 8–13, 2020, Virtual Event, USA.* ACM, New York, NY, USA, 11 pages. https://doi.org/10.1145/3368089.3409768

1 INTRODUCTION

Incidents, or unplanned interruptions or outages of the service, are inevitable in large and complex online service systems [7, 8, 10, 16]. In practice, system failures keep occurring due to frequent updates of system components, changes in operation environment, mobility of devices, etc. Some devastating incidents in large online service systems can significantly degrade system availability and drastically affect user experiences. For example, the estimated cost of an one-hour downtime for Amazon.com on Prime Day in 2018 (its biggest sale event of the year) is up to $100 million[1]. Therefore, many cloud service systems build the incident management (IcM) system to manage incidents and ensure high quality services [7, 23].

When an incident occurs, on-call engineers (OCE) will check the system logs and conduct troubleshooting actions [23]. Meanwhile, engineers will sometimes link several inter-related incidents for efficient and effective diagnosis. For example, a few incidents are usually reported together due to system dependencies, where

*Work done during internship at Microsoft Research Asia.
†Corresponding author

ESEC/FSE '20, November 8–13, 2020, Virtual Event, USA

ACM ISBN 978-1-4503-7043-1/20/11…$15.00
https://doi.org/10.1145/3368089.3409768

[1]https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

mitigating one incident can well resolve several other incidents. Moreover, the links may represent other relevant information such as re-occurrence relation or co-occurrence relation between two incidents.

Detecting and recording such related incidents are valuable for incident diagnosis. Large online systems are composed of many services, with these services coupling with each other heavily. The incidents may happen in one service and are actually caused by problems in another service. Incident links discovered by OCEs can provide strong evidence on such coupling structure and bring insights to fault diagnosis.

Some traditional techniques used for duplicate bug detection can partially be applied to the incident linkage problem in online service systems [9, 29, 33]. For example, natural language processing techniques used for mining textual information in bug reports can help us retrieve similar bug reports [29, 33]. Extending information retrieval framework with textual information is more effective in locating similar bug reports [38]. However, linking incidents of large-scale online service systems remains a complicated problem and contains several new challenges:

- First, incidents are not only manually written by reporters, but are also generated automatically by monitors [7]. The hybrid reporting interface in IcM system provides a convenient and reasonable tunnel for collecting useful information for investigating the incident, but these reports are intertwined with different patterns of textual description, while many traditional bug duplicate methods only focus on human-written bug reports [28, 40].
- Second, incidents with distinct descriptions may be linked either because they share the same cause or they are sequentially happened. For example, a failure in the physical network will cause the failure of the virtual network, and both of these two failures will be reported as incidents. Mitigating the source incident will automatically mitigate all the other incidents caused by this incident. However, traditional text-based duplicate bug detection methods will label these two failures as different incidents [21]. In practice, OCEs tend to link two incidents together for understanding and fixing the problems.
- Third, the dependency structure is neither known nor fixed in large online service systems: the inter-dependency among different services and components could be highly complex, and this inter-dependency keeps changing as there are frequent updates in the system. Therefore, it is hard to use the dependency graph to identify linked incidents as done in [30]. To our knowledge, no previous work has been conducted on such a link identification problem with component dependencies, which is especially important for large online service systems.

In this paper, we investigate the underlying links among the incidents in large online service systems. We propose an effective framework for predicting linked incidents, which is called **LiDAR** (Linked Incident identification with DAta-driven Representation). **LiDAR** is a scalable framework that can identify possible links among a large number of incidents using both semantic information in incident description and the dependency structure

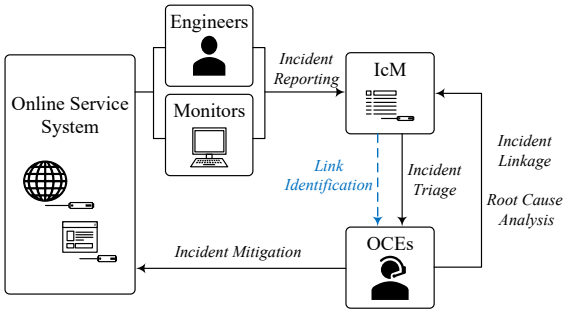

Figure 1: The workflow of an incident management system in Microsoft.

of the online service system. For the semantic information, we use a deep textual embedding module to learn the semantic information, which is able to consistently encode the same symptom reported from both automated monitors (expressed in a structured pattern) and human engineers (expressed in a natural language). Meanwhile, we use a representation learning approach to capture the inter-dependency relationship among different components. At last, we learn the links among incidents by combining both the incident semantics and the service dependency representation.

Our major contributions are as follows:

(1) We investigate the linked incidents problem in an incident management system of large online service system.
(2) We construct a component dependency network and adopt a network representation learning technique to encode the components into a latent feature space. The component dependency network is extracted from historical incident links and is embedded to preserve the network structural information.
(3) We encode the textual description of incidents with an advanced deep learning model to capture the semantic features for incident link identification.
(4) By utilizing the inter-dependency among components as well as the textual information in the descriptions of incidents, we develop an integrated framework that can effectively identify possible linked incidents.

The remainder of this paper is as follows: an overview on the linked incidents of a large online service system in Microsoft is shown in Section 2. In Section 3, we introduce our proposed framework for incident link identification in IcM systems for large online service systems. Experimental results of our method on a real-world IcM system and an extensive case study are shown in Section 4. We discuss lessons learned in Section 5, summarize related work in Section 6, and conclude our work in Section 7.

2 OVERVIEW OF LINKED INCIDENTS IN AN ICM SYSTEM

2.1 The Incident Management System in Microsoft

The workflow of the investigated IcM system is shown in Fig. 1. To detect incidents, various system monitors and alerting tools are deployed at different levels of a real-world large online service

Table 1: An illustrative of different types of linkage.

Parent Title	Parent Component	Child Title	Child Component	Link Type
Auto-monitoring Database job report has not updated	BBB\CCCDev	Database report was missing since this morning	CCC\CCCInternal	Duplicate
Auto-monitoring Database job report has not updated	AAA\Triage	Multiple devices failed to response at Activity	DDD\Not Many	Sequentially Related
Machine Monitoring system has job failing with system errors (code #50).	EEE\Expert-EN	Alerting from Machine Monitoring system in the Pacific Area	AAA\Triage	Sequentially Related

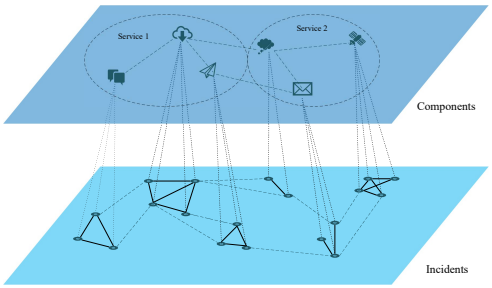

Figure 2: Illustration of the system structure of a large online service system.

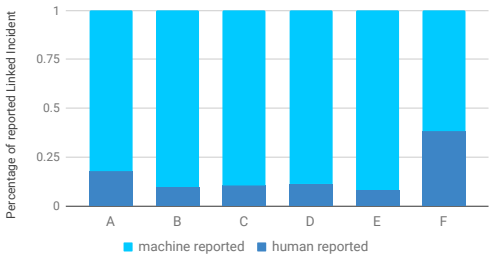

Figure 3: Percentage of human and machine reported linked incidents in different applications (Application A-F)

system in Microsoft. As shown in Fig. 2, the system is constructed with a multi-layer structure.

Microsoft provides thousands of online services for billions of users all around the world. In this section, we first provide an overview of the incident management process in a large online service system from Microsoft and discuss the potential of using incident linkage information to accelerate incident mitigation and root cause analysis. Then, we conduct a detailed investigation on incident links, such as conditions when incidents are linked and examples of linked incidents. Finally, we will introduce the hierarchical structure of the system, and show the importance of correctly identifying cross-component incident links. Due to privacy policies of Microsoft, we have deliberately masked sensitive data throughout this paper.

In the IcM system, each application is composed of several services and each service is composed of several components. Each component will detect if an element fails to meet its expected functionality. If an element fails, an incident report will be automatically generated using the pre-defined patterns filled with the corresponding environment variables, and will be reported to the IcM system. Meanwhile, the IcM system usually allows its users to manually submit incident reports with customized descriptions to the system, because there are always failure situations that have impact on customers but cannot be detected by existing monitors.

The creation of an incident in the IcM system would immediately trigger the triage process, where engineers in the IcM system would investigate and find the correct responsible component to take over the incidents. Once the responsible component of the incident is found, OCEs within the assigned component would try different actions to mitigate the incident. After the incident is successfully mitigated, the engineers in the team will conduct postmortem analysis on the root cause of this incident. They will also determine whether this incident should be linked to other existing incidents, and label all the linked incidents accordingly. This is also known as the incident linkage step.

Through the whole process, the incident mitigation step usually consumes a lot of time and labor resources. This consumption can be greatly reduced by transferring past knowledge and solutions to current incidents by identifying the links. Therefore, this work focuses on identifying incident links for OCEs (shown as the dashed line in Figure 1) so as to quickly find the mitigation solution as well as to figure out the right root cause of the incident.

2.2 Incident Links

In the IcM system, there are a variety of incidents linked by on-call engineers. We illustrate some incident links in Table 1. In general, incidents are linked due to duplicate descriptions or relatedness.

2.2.1 Duplicate Links. Duplicate links are those linked incidents that refer to the same malfunction case described from the same or different perspectives. For example, the first line in Table 1 is a typical duplicate link. As we have mentioned in the previous section, incidents are reported by both human and online monitors, the linguistic style of whose descriptions for the same incident may be quite different.

The first line of Table 1 shows the linked incidents, where the parent and child incident title are similar, but with different styles of description. In our system, parent incidents refer to incidents investigated by engineers earlier , while child refers to those investigated later. The parent incident title starts with "Auto-monitoring", which indicates that the report is sent by a monitor automatically.

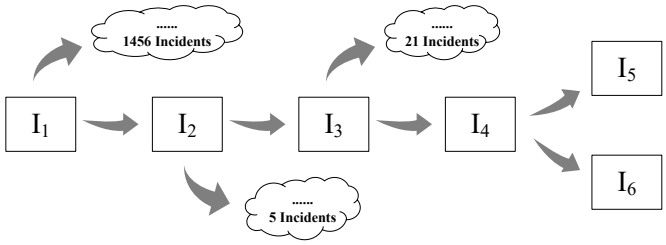

Figure 4: An illustration of linked incidents.

While the child incident is a simple description about a missing database report, which is probably reported by a human engineer. Further, these two incidents happened at different components mostly because their reporting sources are from different services. After investigation, it has been found that both of the two incidents describe the database missing report issue, but they are described in different ways and come from different aspects of the system. As shown in Figure 3, for the six applications in our IcM system, at least around 10% of the linked incidents are related to human reported incidents. Therefore, proposing approaches that can capture both human and machine reported incidents is helpful for finding mitigation methods and root cause analysis.

2.2.2 Sequentially Related Links. The second type of link is when the incidents happen sequentially. In large online service systems, different components work jointly to support a number of online services. It is common that one incident that indicates the malfunction of some parts of the system may also affect the functionality of other parts, which leads to incidents that happen sequentially. For example, if the storage of a cloud system broke down, many products running on cloud storage would probably be impacted. If we treat these incidents independently, it will largely ignore the rich information of the functional inter-dependencies [39, 42, 43]. In practice, linking these incidents together can also significantly help engineers understand specific incidents and better fix the incidents.

Figure 4 gives an illustrative example of a series of sequentially reported incidents in the investigated large online system. In Figure 4, I_1 is an outage with regard to the cooling condition in a data center, which is also an incident with high impact. It directly causes a great number of incidents. Among them, I_2 is also an impactful incident related to the cooling system, and it causes six other incidents. Similarly, I_3 is caused by I_2, and causes 22 incidents including I_4. Finally, incident I_4 causes two alertings: I_5 and I_6. This type of linked incident series is quite common in the system, which may substantially reflect the possible operational and functional dependency on different components, services and projects.

Among all incidents presented in Figure 4, the textual description of these incidents may be distinct, and the components these linked incidents belong to can be quite different. All these incidents happened sequentially (one after another). Traditional bug de-duplicate methods have not considered this type of links.

2.3 Cross-Component Links

In the current IcM system, a great proportion of the incident links are cross-component links, meaning that the two linked incidents belong to different components, which may be either duplicate links or related links. These links are strong indicators of the underlying component structure, which keeps evolving due to frequent updates in the system. For example, in the following case, *Component A* and *Component B* are related by the sequential link between *Incident A* and *Incident B*. Such cases of cross-component incident links occur frequently in our investigated system. Previous bug de-duplicate methods mainly focus on incidents happening at the same component and fail to deal with this situation. In our attempt to predict incident links, we consider characterizing this complex component dependency structure in addition to utilizing the textual description of incidents.

Table 2: Title of the incidents in Figure 4

Incident	Title
I1	Cooling system is malfunctioned and impacted multiple services in the SA area
I2	SA area Datacenter Experiencing Cooling Issues
I3	Alertings reaches threshold, Temperature Metric Api is overload
I4	Database Account Login Error and the dependency components Error
I5	Alerting: dependency component analysis meets errors
I6	Alerting: Web Api has delays.

Incident A: Application A encountered a network outage.
Component A: Network Controller
————————–
Incident B: Storage timeouts in Region WU.
Component B: Web Service Team

Overall, incident linking is both a challenging and important function in an IcM system. To correctly predict true links among incidents, we not only need to effectively learn the semantic representation of a sentence, but also need to model the system component dependency structure with high quality. In our work, we consider both semantic and component dependency features to learn incident links.

3 PROPOSED APPROACH

To deal with different types of incident links, we propose **LiDAR**, a deep learning based model for identifying incident links. In this section, we will first introduce the overall structure of the proposed model, which is composed of two modules dealing with the textual descriptions and the component dependency information, respectively. Then we will describe each module in detail. Finally, the integration combining these two modules for link identification will be described.

3.1 An Overview

The overall structure of the proposed method, LiDAR, is shown in Figure 5. To consider both the textual information and system structure information, we extract two features from each incident for learning: the description of incident in natural language and the corresponding component that is responsible for the incident.

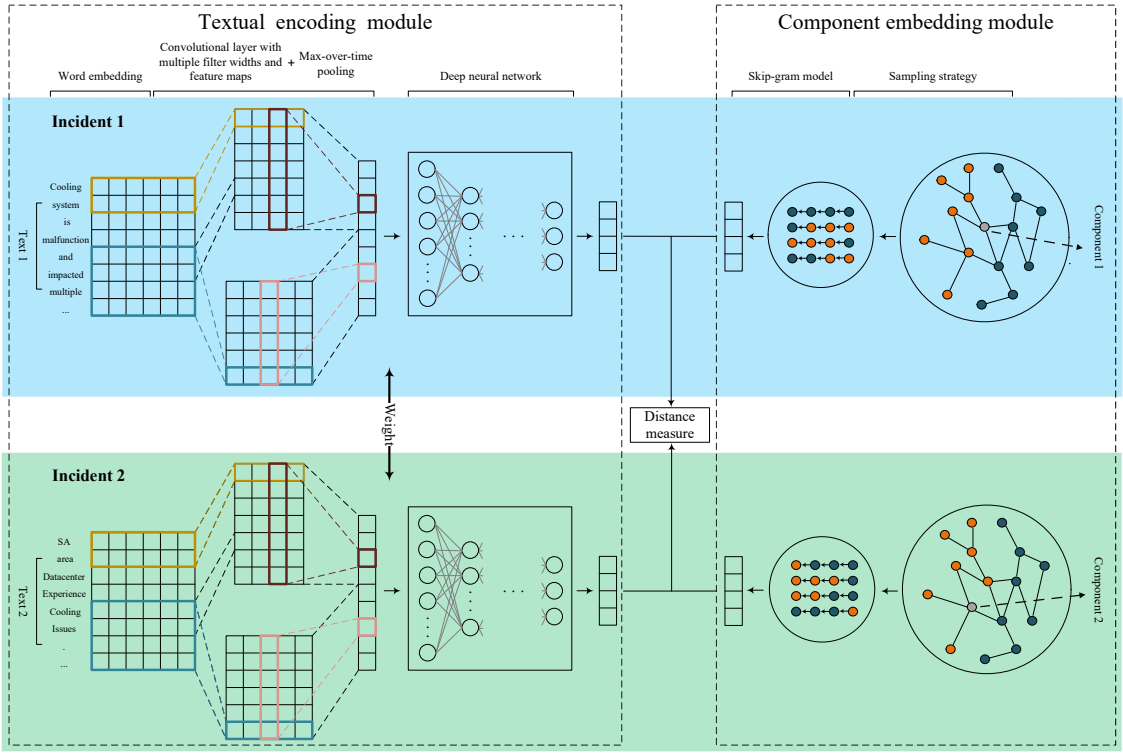

Figure 5: The overview framework of LiDAR. The model mainly consists of two modules: textual embedding module and component representation learning module.

These two features are fed into two modules: the textual embedding module and the component representation module. These two modules are aimed to extract the textual information and the component dependency information, respectively.

First, to deal with the challenges brought by textual description from different sources, we introduce a text encoder that can grasp the semantic representation of the incident description rather than textual only information. The textual module is trained to map the textual description into a high dimensional semantic representation.

Then, to understand the system architecture and dependencies, component representation module is realized with a novel network representation learning approach. Each component in the system is learned to be embedded into a low-dimensional latent space according to their correlation with other components according to historical links. Afterwards, the component-level similarity is obtained by the similarity of these component embedded vectors, capturing both the similarity in structural position as well as the graph distance between two components.

After calculating similarities from these two distinct modules, we integrate them together to obtain the final score and set a threshold for link identification.

3.2 Modeling Textual Description

An incident title describes the symptom of the incident, including the location, detailed occurring environment, impacted services and customers, etc. This kind of comprehensive information is of great help in finding potential links with other incidents. Here we learn the relationship between two incidents' descriptions by adopting a representation approach using deep neural network.

The module for textual information extraction is a representation learning process in nature, which learns a mapping from texts in natural language into a representative numerical vector in the latent space, and pushes the linked incidents close to each other in this space. To better extract the semantic features, we use a three-stage textual representation method, including word-level embedding, deep feature mapping, and semantic comprehension. Specifically, word-level embedding is used for learning textual information, while the other two stages aim at learning higher level semantic features for each title.

In the first stage, the textual description in natural language is embedded word-by-word as word vectors using *FastText* [25]. This method provides a scalable solution to generate pre-trained word vectors on infrequent text by using sub-word vectors. Each incident title is then encoded as a numerical matrix of dimension $L \times D$ where L is the number of effective words in the description,

and D is the length of the vector for each word, which is also the dimension we want to preserve in this encoding stage.

In the second stage, multiple 1-D convolutional kernels and a max-over-time pooling layer are applied on the matrix obtained from the first stage. With the application of multiple convolution kernels of different sizes, we can get multiple feature maps extracting the correlation among the words in different lengths of n-grams. The max-over-time pooling layer selects the most representative n-grams in each channel, and thus provides a succinct representation for all the meaningful n-grams of the description.

In the third stage, a neural network with fully-connected layers is adopted to learn the useful relations among the representative n-grams, and outputs the learned representative vector for the description of each incident.

To learn the matching degree between the descriptions of incidents, this encoding module is trained based on a Siamese neural network [20], which is a state-of-the-art method for learning entity matching tasks. It consists of two twin neural networks, one for each link description in a pair of incidents with known relation. These two neural networks are identical in structure, sharing the same configuration with the same parameters and weights [18]. This kind of structure focuses on learning embeddings in deep layer that places the same descriptions closer, and can learn underlying semantics. With the help of such a Siamese neural network, the encoding schemes are well-learned in a supervised fashion according to the incident link labels.

3.3 Modeling Component Dependency

The component dependency structure is learned by treating the components and their dependency as a network. Historical links are used to inspect component relations, and these components are then learned to be represented by a low dimensional vector retaining their structural information in the dependency network.

3.3.1 Construction of Component Dependency Network. In large online service systems, a component is usually responsible for a certain kind of incidents. For example, the database component is likely to encounter incidents such as running out of memory, no response to query, etc. To some extent, component information provides strong hints on possible links over the incidents.

The component dependency network is constructed in the following way: For all the historical incident links up to a specific time point, we find all the corresponding components of the involved incidents. Then for each pair of different components, connect them with an edge if they each have at least one incident linked to each other. A component dependency network $\mathbf{G} = (\mathbf{V}, \mathbf{E})$ is thus constructed with $\mathbf{V} = \{v_i\}$ representing the set of components, and $\mathbf{E}$ representing the set of existence of historical incident links between every two components.

3.3.2 Encoding Component Dependency Network. The component dependency will be encoded into a network representation, which is in essence a mapping function from all the component $\mathbf{V}$ in the component dependency network $\mathbf{G}$ to a continuous $d-$dimensional feature space g. The primary challenge in this domain is finding a way to represent, or encode, the network structure so that it can be easily exploited by machine learning models. Here we utilize *node2vec* [14], an algorithmic framework for learning continuous feature representations for nodes in networks. Based on the established component dependency network, we first adopt the 2-nd order random walk sampling strategy in *node2vec* and obtain sequences of components. Then, the skip-gram representation learning framework [26] is adopted to encode components as dense low-dimensional representations.

Finally, the distance (or similarity) between two given components in the dependency network can be easily calculated based on the output of the component representation learning module.

3.4 Combining Structural and Textual Information

Both the textual embedding and the component representation learning will output its own vector representation for different types of information. Based on these learned representations, we can calculate the linkage confidence score $s(i, j)$ for any two incidents i and j:

$$s(i, j) = \alpha * s_{\text{comp}}(i, j) + (1 - \alpha) * s_{\text{text}}(i, j) \tag{1}$$

where s_{comp} and s_{text} represent the similarity calculated using the vector representations from the component representation learning module and the textual embedding module respectively. Here the similarity score s_{comp} and s_{text} is calculated by the cosine distance in corresponding semantic space. The larger $s(i, j)$ is, the more likely that there exists a link between incidents i and j. In practice, we can identify links $\ell(i, j)$ by a threshold γ cutting the score s into a binary response:

$$\ell(i, j) = \begin{cases} 1 & \text{if } s > \gamma \\ 0 & \text{otherwise.} \end{cases} \tag{2}$$

This threshold parameter γ in (2) and the weight parameter α in (1) can be chosen according to the prediction performance on a separate validation set. The validation set is selected as 5% incident links from the training set.

4 EXPERIMENTS

4.1 Dataset and Setup

4.1.1 Dataset. Our proposed incident linking framework is evaluated in the large online service system investigated in Section 2. We collect all the linked incidents from 465 different services, including 1100 components in the evaluated IcM system. We split the training and testing set by the reported time of incident links. More specifically, we build the training set with incident links from January 01, 2017 to August 31, 2018 and the testing set with links from September 01, 2018 to October 31, 2018.

4.1.2 Setup. In the component representation learning module, the component dependency network is established using only training data. We set the embedded dimension $d_C = 64$, walks per node $r = 10$, walk length $l = 20$, and walking parameters $p = 2.3, q = 1$. In the textual embedding module, the convolutional layer uses three sets of convolution kernels with 3 different widths (3, 4 and 5), each of which has 100 kernels. The preserved word embedding dimension is 300, and the number of training epochs is 30.

Table 3: Link identification results

		P1	P2	P3	P4	P5	P6	P7	P8	P9	P10
DWEN	F1	0.9271	0.7288	0.7963	0.9922	0.6745	0.6455	0.9314	0.7372	0.6133	0.8765
	Precision	0.9145	0.7500	0.7679	0.9958	0.5115	0.6171	0.9392	0.6914	0.7617	0.8899
	Recall	0.9400	0.7087	0.8269	0.9886	0.9899	0.6766	0.9237	0.7894	0.5133	0.8635
DBTM	F1	0.9660	0.7201	0.7869	0.9468	0.9816	0.4472	0.9217	0.7754	0.2854	0.9398
	Precision	0.9948	0.9441	0.8215	0.9992	0.9947	0.7940	0.9852	0.9768	0.9209	0.9610
	Recall	0.9388	0.5802	0.7551	0.8996	0.9947	0.3112	0.8658	0.6428	0.1688	0.9195
Simple	F1	0.8164	0.6944	0.7806	0.8202	0.5854	0.5268	0.8091	0.8077	0.6755	0.8036
	Precision	0.8474	0.8026	0.7246	0.8826	0.7273	0.6778	0.7749	0.7434	0.5466	0.7763
	Recall	0.7876	0.6119	0.8460	0.7660	0.4898	0.4308	0.8465	0.8842	0.8839	0.8329
LiDAR-T	F1	0.9417	0.8031	0.9378	0.9958	0.9735	0.6756	0.9697	0.9168	0.6838	0.9350
	Precision	0.9408	0.7365	0.9229	0.9993	0.9892	0.5411	0.9923	0.9377	0.6045	0.9756
	Recall	0.9426	0.8829	0.9532	0.9923	0.9583	0.8991	0.9481	0.8968	0.7870	0.8976
LiDAR-C	F1	0.9888	0.9334	0.9153	0.9986	0.7253	0.8772	0.9848	0.9795	0.7677	0.9166
	Precision	0.9951	0.9272	0.8735	0.9982	0.5690	0.7815	0.9806	0.9738	0.6583	0.8503
	Recal	0.9826	0.9397	0.9613	0.9990	0.9998	0.9996	0.9890	0.9852	0.9207	0.9941
LiDAR	F1	0.9890	0.9495	0.9512	0.9988	0.9924	0.8877	0.9877	0.9820	0.7698	0.9397
	Precision	0.9995	0.9647	0.9528	0.9988	0.9849	0.8725	0.9858	0.9773	0.6646	0.9553
	Recall	0.9787	0.9348	0.9496	0.9988	0.9999	0.9034	0.9896	0.9867	0.9145	0.9246

Our study is conducted on Ubuntu 16.04 with 24-core Dual-Intel Xeon E5-2690 v3 CPU (2.60GHz),220 GB memory, and a single NVIDIA Tesla K80 GPU accelerator.

4.2 Incident Linkage

To evaluate the performance of the proposed incident linkage method, *LiDAR*, we test our method on ten ($P_1, P_2, ..., P_{10}$) applications in Microsoft, and adopt several classical metrics for evaluating classification tasks.

In the experiment, **LiDAR** is compared with the following state-of-the-art and baseline methods:

(1) **DWEN[3]**: a deep learning based duplicate bug report detection method, which extracts text features from bug reports and builds a neural network model for predicting whether or not two bug reports are duplicate.
(2) **DBTM[27]**: an information retrieval based duplicate bug report detection method, which models a bug report as a textual document describing certain technical issue(s), and models duplicate bug reports as the ones about the same technical issue(s). It takes advantage of both IR-based features and topic-based features.
(3) **Simple**: a naive approach which uses the historical linked components for identifying whether two incidents belongs to the same component, if they belong to the same component then these two incidents are linked.
(4) **LiDAR-T**: This is the textual part of our proposed method. Only textual information is used for prediction.
(5) **LiDAR-C**: This is the structure part of our proposed method, only component representation information is used for prediction.
(6) **LiDAR**: This is our proposed method.

The comparison results are shown in Table 3. The ten subject applications are labeled as $P_1, P_2, ..., P_{10}$ in the table. It is evident that our method can outperform other state-of-the-art duplicate bug detection methods in the incident linkage task among all ten applications. Moreover, except for *LiDAR*, DWEN has obtained the highest precision among all other comparing methods in all ten different applications we tested. The information retrieval based method DBTM has low recall values compared with other methods. This is mainly because reports for the same incident could use different descriptions in our IcM system, but the DBTM method is mostly text based. If some linked incidents do not have the same words in their descriptions, the text-based methods could fail to detect them. Meanwhile, comparing to other baseline methods, alternative methods *LiDAR-T* and *LiDAR-C* can obtain a significant performance improvements in all subject applications. We can further find that the overall performance of *LiDAR* is better than *LiDAR-T* and *LiDAR-C*, because combining the component information and textual information can compensate each other and provide better performance.

4.3 Human-Machine Incident Linkage

An IcM system contains incidents reports from two sources: human and machine. Incidents from different sources use quite different writing styles.

Although most of the reported links are between machine reported incidents *(machine-machine links)*, practitioners pay great attention to investigating links between human reported incidents and machine reported ones *(human-machine links)* to improve the system [6]. Many text-based duplicate bug report detection methods, such as DBTM [27], fail to deal with this scenario. Our proposed

Table 4: Results on human-machine and machine-machine links. (H-M Links represent human-machine links, M-M Links represent machine-machine links.)

	Link Type	F1	Precision
DWEN	H-M Links	0.6043	0.8370
	M-M Links	0.9756	0.9986
LiDAR	H-M Links	0.7584	0.8022
	M-M Links	0.9981	0.9975

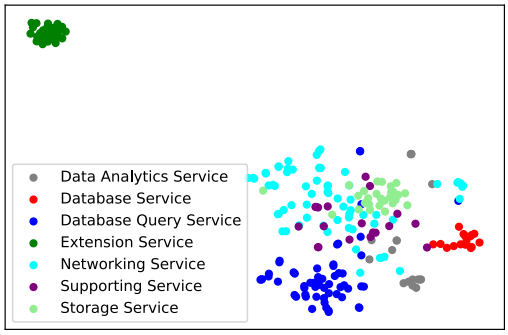

Figure 6: Visualization of embedded component vectors from 7 services.

approach is capable of identifying both types of links by incorporating different sources of information.

To evaluate the performance for human-machine links, we randomly select 10,000 human-machine links from our IcM system. Meanwhile, we select 10,000 machine-machine links in comparison. The results are listed in Table 4. As confirmed by previous empirical studies [7], deep learning based methods can obtain the best performance among all other methods. Here we list both *LiDAR* and DWEN's performance for human-machine link prediction task. Due to the space limit, we only listed the DWEN baseline results in Table 4. According to Table 4, both *LiDAR* and DWEN can obtain good performance in machine reported incident linkages, over 97 percent to be specific. This is because machine reported incidents share almost the same template and descriptive patterns, which can easily be learned by both models. However, for the human reported incidents, it is hard to learn the descriptive patterns. The *LiDAR* method is able to understand the semantics of incident descriptions other than the textual information, and thus can help link the human reported incidents better. As a result, *LiDAR* obtains a significant performance improvement comparing to DWEN in the human-machine link prediction task.

4.4 Effectiveness of Component Representation Learning

Apart from the performance evaluation we have shown in Table 3, we have also investigated the effectiveness of component representation learning. Ideally, the encoded component vector should give a good representation of the component. One way to check the effectiveness of learned representation is to investigate whether the component vectors from the same service can be grouped together.

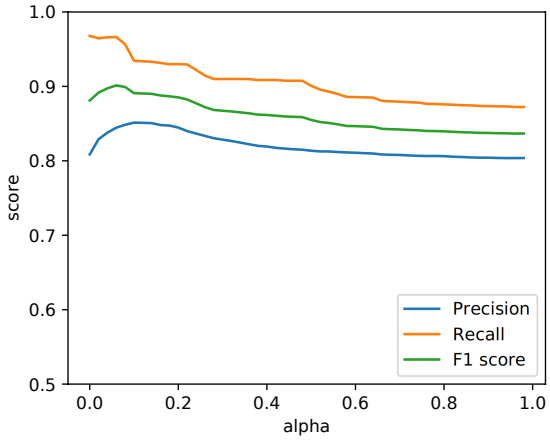

Figure 7: Sensitivity over hyper-parameter α.

Figure 6 visualizes multiple embedded component vectors with dimension $d_C = 64$ from the following 7 services: Data Analytics Service, Database Service, Database Query Service, Extension Service, Networking Service, Support Service and Storage Service. To visualize these 64-d vectors, we apply the t-SNE (t-distributed stochastic neighbor embedding) method [24] to transform the vector from the high-dimensional space into a two-dimensional space. In Figure 6, components from the same service are presented in the same colour. By looking at the spatial distribution of components, we can find that our embedding method can cluster the components from the same service together in the latent space. This observation reflects the effectiveness of component representation learning: similar components (from the same service) have similar encoded vectors and are more likely to have incident links.

4.5 Parameter Sensitivity

In our proposed approach, the combination of textual embedding model and component representation learning model is conducted by an ensemble step with the parameter α in Equation (1). To investigate the sensitivity of model performance on the value of this parameter, we examine the performance of our proposed model under different α values.

Figure 7 records the values of three performance metrics (i.e., precision, recall and F-1 score) obtained in the link identification task with the value of α varying from 0 to 1. We can see that all of these three evaluation metrics remain stable under different parameter settings. Therefore, we can conclude that LiDAR is parameter insensitive from this experiment, and thus it is easy to use without huge effort in tuning the weight parameter.

4.6 Case Study

To evaluate the effectiveness of LiDAR in practice, an extensive case study is conducted. It turns out that the proposed method can discover the links between incidents in the large online service system we discussed in Section 2. A virtual machine reboot case is chosen for illustration. One day, a population of customers had experienced the virtual machine reboot issue in the CN area. According to the diagnosis records in the IcM system, this incident

Table 5: A list of candidate incidents that would be potentially linked to an incident happened in CN area

Ranking	Component	Incident Title	Correctness
1	Cloud Service Cache	CN component is unhealthy based on on-call testing	✓
2	Service Near Real Time Alerting	Detected Error: Suspicious Application A Enumberate	
3	Storage Engineering	Customer impacted in the CN area for the Storage issues	✓
4	Cloud Service Cache	Highlight: CN area is unhelthy because ignored storage alerts	✓
5	Storage Triage	CN component is unhealthy based on on-call testing	✓
6	Cloud Service Cache	cluster is unhealthy, and runners and organic counters are having issues	✓
7	Storage Triage	Hit limit for overall Bandwidth/TPS stamp Throttling on some cluster	
8	Storage Triage	Storage Roles Quorum Check failed Severely from cluster	✓
9	Cloud Service Cache	CN component is unhealthy based on on-call testing.	✓
10	Storage Auto Trouble Shooting	Partition does not seem to have been loaded from Primary cluster	
11	Cloud Service Cache	CN component is unhealthy based on on-call testing	✓
12	Service Near Real Time Alerting	Application Credential or Service Principal changed	
13	Storage Triage	Storage staffs check failed from a CMCluster.	✓
14	Storage Database Table	IOM has detected customer impacting incident for Storage in CM Region	✓
15	Storage Triage	Storage Roles Quorum Check failed Severely from CMCluster	✓
16	Cloud Service Cache	CN component is unhealthy based on on-call testing.	✓

is caused by a network failure. To investigate which incidents are related to this incident, we collect hundreds of incidents happened around the reported time of this investigated incident.

The results of identified incidents that are likely to be linked to the investigated one is shown in Table 5. We list the top 16 most probable candidates ordered by the calculated linking score using LiDAR. Having checked with on-call engineers, we find that 12 out of these 16 incident candidates have links with the investigated incident (related to the virtual machine rebooting issue). As shown in the table, top incidents happened at the Cloud Service Cache and Storage components. These components are recognized to be commonly affected by network issues: the Cloud Service Cache depends heavily on network errors, while the performance of storage is largely influenced by the network as well. As previously shown, the root cause of the investigated incident is a network failure. Thus, it is quite reasonable to find that incidents from both the Cloud Service Cache component and Storage component are more likely to be linked with the investigated incident.

Meanwhile, incidents' textual descriptions imply rich information. Many incidents that are linked to the investigated incident contain the keyword CN area. This keyword gives the occurring location information of the incident, which is obviously an important indicator of possible links. Therefore, our model can automatically capture this important textual information for link identification. Another keyword that many linked incidents have is 'Storage'. This observation is consistent with the findings in the above paragraph that the Storage component plays an important role.

5 LESSONS LEARNED

Modeling semantics information for incidents. In our investigated system, incidents are reported by both engineers and automated monitors. Different incident formatting patterns from human and machine lead to different textual distribution in the incident titles, although the semantic information they try to convey is quite similar. To learn the linking patterns from diverse textual sources, we adopt the siamese network based on textual convolutional neural network (TextCNN), which is proved practical for mining semantics of natural language [18]. By virtue of the TextCNN framework, our model can not only learn links between incidents that are similar in textual descriptions, but also those incidents with different textual descriptions but the same semantic meaning, i.e., the duplicate incidents from different sources.

Effectiveness of Components Representation. The underlying component dependency structure of online service systems is extremely complex. Although a complete component structure can be obtained by piecing up the designed system architecture, the actual functional dependency and common error propagation structure can hardly be derived purely from these system architectures. Therefore, we choose a data-driven approach to establish such a component dependency graph from an incident perspective. The learned component dependency graph is therefore more appropriate for analysis and diagnosis of incidents, and can help us understand such a large complex system better.

Fusing information from different sources. As explained, both the textual description and the component structure contain key information about different aspects of incident links, which motivates us to combine these two sources together in our model. The TextCNN module can well capture the underlying semantic information in the incident title, and the siamese network is capable of learning to match incidents of all types of links. Moreover, the learned component representation naturally benefits the identification of links across components, including most duplicate links and related links, and thus can supplement and improve the incident linking model. By integrating different data types and different information sources, our proposed model sees a great improvement in the incident linking task.

6 RELATED WORK

6.1 Duplicate Bug Retrieval

There have been a number of work in literature investigating the problem of duplicate bug detection and retrieval. Techniques from

information retrieval were adopted in early days, focusing on measuring the similarity between newly reported bugs and existing bug reports. Nature language processing methods have been used to capture the textual information in the bug reports.

[29] evaluated the feasibility of using Natural Language Processing techniques to help with automate detection of duplicate defect reports. However, this approach did not consider other categorical information of bugs such as component and priority. [34] combined execution info with natural language info to detect duplicate bug reports. [33] leveraged support vector machines (SVM) to train a discriminative model for duplicate bug report retrieval. [32] extended a textual similarity measure called BM25F for lengthy structured reports, and proposed a retrieval function to measure the similarity between two bug reports. [19] extended previous work by introducing a range of metrics based on the topic distribution of the issue reports. [38] combined a traditional information retrieval technique and a word embedding technique for duplicate bug detection, taking bug titles, descriptions, as well as bug product/component information into consideration. [27] modeled a bug report as a textual document describing certain technical issue(s) and duplicate bug reports as the ones about the same technical issue(s), and used this model to detect duplicate bugs. [15] investigated how contextual information about software-quality attributes, software-architecture terms, and system-development topics can be exploited to improve bug de-duplication. This work replicated the method in [32] and extended it by adding contextual comparison, through the addition of contextual features to the bug reports. [41] identified textual and statistical features of bug reports and applied learning to rank techniques to detect duplicate bug reports.

Recently, deep learning models have shown superior performance to classical machine learning approaches in many tasks. Therefore, some works have applied deep learning techniques to improve the accuracy of duplicate/similar bug detection tasks [37]. [11] proposed a retrieval and classification model using Siamese Convolutional Neural Networks (CNN) and Long Short Term Memory (LSTM) for accurate detection and retrieval of duplicate/similar bugs. [3] trained a deep neural network with word embedding for duplicate bug report detection in software repositories. [4] combined Latent Dirichlet Allocation (LDA) and word embedding for duplicate bug report detection.

6.2 Fault Diagnosis of Software-Intensive Systems

Our approach for incident linking could largely help with the process of root cause analysis, which is also a well-studied topic in literature. Root cause analysis, also referred to as fault localization, fault isolation, or alarm/event correlation, is the process of inferring the set of faults that generate a given set of symptoms [17]. It is the key procedure of troubleshooting in large systems [2], and is widely applied in a number of different areas, such as computer networks [12, 22], software [1, 35], industrial systems [5, 13] and so on. Understanding the root cause of observed symptoms can largely help the healing and improvement of large online systems. [31] reviewed different techniques in the literature and how they satisfy performance and scalability requirements. It also listed several types of challenges with regard to system-level RCA, including the combination of both domain knowledge and system knowledge.

There are also works on failure prediction for a system with inter-connected components [39, 42, 43] in literature, but few work has focused on predicting the potential links among the numerous incidents. [36] proposed an approach towards automated repair of software-defined networks (SDN), using the causal graph (called meta provenance). However, this kind of approach cannot be directly applied to complex online systems for two reasons: first, the incidents/failures in SDN applications are restricted in smaller domains, while in large online systems the incidents come from much more types of sources; second, the incidents in SDN applications can form a much denser causal graph, but the incidents in large online systems tend to spread into many different components, making their relations quite sparse in terms of an incident graph.

7 CONCLUSION

The identification of linked incidents is important for building an intelligent IcM system for large online service systems. Correctly identifying linked incidents can not only help mitigate the incidents, but also analyze the root causes to prevent recurrence of similar incidents. In this paper, we investigate the linked incidents in a large online service system, and find two important indicators for identifying potential links among numerous incidents: the component dependency learned from past incident links and the textual description of incidents. Starting from this observation, LiDAR, a deep learning based framework integrating two representation learning modules (for the component dependency and the textual descriptions), is designed to discover potential links among numerous incidents. Extensive experiments and studies have shown the effectiveness and applicability of our proposed method. We believe our work can greatly facilitate incident management for large-scale online systems.

Despite the superior performance of LiDAR, there are limitations in current implementation. First, the component dependency network is established using historical links, thus it does not consider new components that have not appeared in the historical incidents. Moreover, the training phase of current implementation is done offline, so it does not update in a real-time manner.

In the future, more effort can be devoted to the following directions: 1) Improving the link identification performance by incorporating more data sources, such as the time and location of incident occurrence, severity of incidents; 2) Developing an online learning scheme based on the current work so that any new signals on incident linkage can be integrated into the model in time; 3) Extending the current work to combine with other adjacent processes such as the triage phase and the root cause analysis process to make the IcM system more intelligent.

ACKNOWLEDGMENTS

We thank our colleagues at Microsoft Azure groups who developed the incident management system and helped us learn the system: Feng Gao, Jeffery Sun, Pochian Lee, Li Yang, Zhangwei Xu. This work is supported by the National Natural Science Foundation of China (NSFC) under Grant No. 61702107. Hongyu Zhang's work is supported by Australian Research Council (ARC) DP200102940.

REFERENCES

[1] Pragya Agarwal and Arun Prakash Agrawal. 2014. Fault-localization techniques for software systems: A literature review. *ACM SIGSOFT Software Engineering Notes* 39, 5 (2014), 1–8.

[2] Mona Attariyan, Michael Chow, and Jason Flinn. 2012. X-ray: Automating Root-Cause Diagnosis of Performance Anomalies in Production Software.. In *OSDI*, Vol. 12. 307–320.

[3] Amar Budhiraja, Kartik Dutta, Raghu Reddy, and Manish Shrivastava. 2018. DWEN: deep word embedding network for duplicate bug report detection in software repositories. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 193–194.

[4] Amar Budhiraja, Raghu Reddy, and Manish Shrivastava. 2018. LWE: LDA refined word embeddings for duplicate bug report detection. In *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*. ACM, 165–166.

[5] Carlo Cecati. 2015. A survey of fault diagnosis and fault-tolerant techniquesâĂŤPart II: Fault diagnosis with knowledge-based and hybrid/active approaches. *IEEE Transactions on Industrial Electronics* (2015).

[6] Ajay Chandramouly, Big Data Domain Owner, IT Ravindra Narkhede, IT Vijay Mungara, IT Guillermo Rueda, and IT Asoka Diggs. 2013. Reducing Client Incidents through Big Data Predictive Analytics. *Intel IT Big Data Predictive Analytics,(December)* (2013).

[7] Junjie Chen, Xiaoting He, Qingwei Lin, Yong Xu, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. An Empirical Investigation of Incident Triage for Online Service Systems. In *Proceedings of the 41st International Conference on Software Engineering: Software Engineering in Practice* (Montreal, Quebec, Canada) *(ICSE-SEIP '19)*. IEEE Press, 111–120. https://doi.org/10.1109/ICSE-SEIP.2019.00020

[8] Junjie Chen, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2019. Continuous incident triage for large-scale online service systems. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 364–375.

[9] Junjie Chen, Wenxiang Hu, Dan Hao, Yingfei Xiong, Hongyu Zhang, and Lu Zhang. 2019. Static duplicate bug-report identification for compilers. *SCIENTIA SINICA Informationis* 49, 10 (2019), 1283–1298.

[10] Junjie Chen, Shu Zhang, Xiaoting He, Qingwei Lin, Hongyu Zhang, Dan Hao, Yu Kang, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems. In *The 35th IEEE/ACM International Conference on Automated Software Engineering*. to appear.

[11] J. Deshmukh, A. K. M, S. Podder, S. Sengupta, and N. Dubash. 2017. Towards Accurate Duplicate Bug Retrieval Using Deep Learning Techniques. In *2017 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. 115–124. https://doi.org/10.1109/ICSME.2017.69

[12] Lv Feng, Li Xiang, and Wang Xiu-qing. 2013. A survey of intelligent network fault diagnosis technology. In *Control and Decision Conference (CCDC), 2013 25th Chinese*. IEEE, 4874–4879.

[13] Zhiwei Gao, Carlo Cecati, and Steven X Ding. 2015. A survey of fault diagnosis and fault-tolerant techniquesâĂŤPart I: Fault diagnosis with model-based and signal-based approaches. *IEEE Transactions on Industrial Electronics* 62, 6 (2015), 3757–3767.

[14] Aditya Grover and Jure Leskovec. 2016. node2vec: Scalable feature learning for networks. In *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 855–864.

[15] Abram Hindle, Anahita Alipour, and Eleni Stroulia. 2016. A Contextual Approach Towards More Accurate Duplicate Bug Report Detection and Ranking. *Empirical Softw. Engg.* 21, 2 (April 2016), 368–410. https://doi.org/10.1007/s10664-015-9387-3

[16] Jiajun Jiang, Weihai Lu, Junjie Chen, Qingwei Lin, Pu Zhao, Yu Kang, Hongyu Zhang, Yingfei Xiong, Feng Gao, Zhangwei Xu, Yingnong Dang, and Dongmei Zhang. 2020. How to Mitigate the Incident? An Effective Troubleshooting Guide Recommendation Technique for Online Service Systems. In *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Industry track*. to appear.

[17] Soila P Kavulya, Kaustubh Joshi, Felicita Di Giandomenico, and Priya Narasimhan. 2012. Failure diagnosis of complex systems. In *Resilience assessment and evaluation of computing systems*. Springer, 239–261.

[18] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *EMNLP*.

[19] Nathan Klein, Christopher S Corley, and Nicholas A Kraft. 2014. New features for duplicate bug detection. In *Proceedings of the 11th Working Conference on Mining Software Repositories*. ACM, 324–327.

[20] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. 2015. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, Vol. 2. Lille.

[21] Ahmed Lamkanfi, Javier Pérez, and Serge Demeyer. 2013. The eclipse and mozilla defect tracking dataset: a genuine dataset for mining bug information. In *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 203–206.

[22] Ma łgorzata Steinder and Adarshpal S Sethi. 2004. A survey of fault localization techniques in computer networks. *Science of computer programming* 53, 2 (2004), 165–194.

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

[24] Laurens van der Maaten and Geoffrey Hinton. 2008. Visualizing data using t-SNE. *Journal of machine learning research* 9, Nov (2008), 2579–2605.

[25] Tomas Mikolov, Edouard Grave, Piotr Bojanowski, Christian Puhrsch, and Armand Joulin. 2017. Advances in pre-training distributed word representations. *arXiv preprint arXiv:1712.09405* (2017).

[26] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. 2013. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*. 3111–3119.

[27] Anh Tuan Nguyen, Tung Thanh Nguyen, Tien N. Nguyen, David Lo, and Chengnian Sun. 2012. Duplicate Bug Report Detection with a Combination of Information Retrieval and Topic Modeling. In *Proceedings of the 27th IEEE/ACM International Conference on Automated Software Engineering* (Essen, Germany) *(ASE 2012)*. ACM, New York, NY, USA, 70–79. https://doi.org/10.1145/2351676.2351687

[28] Mohamed Sami Rakha, Cor-Paul Bezemer, and Ahmed E Hassan. 2018. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. *IEEE Transactions on Software Engineering* 44, 12 (2018), 1245–1268.

[29] P. Runeson, M. Alexandersson, and O. Nyholm. 2007. Detection of Duplicate Defect Reports Using Natural Language Processing. In *29th International Conference on Software Engineering (ICSE'07)*. 499–510. https://doi.org/10.1109/ICSE.2007.32

[30] Robert J. Sandusky, Les Gasser, and Gabriel Ripoche. 2004. Bug Report Networks: Varieties, Strategies, and Impacts in a F/OSS Development Community. In *Proceedings of the 1st International Workshop on Mining Software Repositories (MSR 2004)*. 80–84.

[31] Marc Solé, Victor Muntés-Mulero, Annie Ibrahim Rana, and Giovani Estrada. 2017. Survey on models and techniques for root-cause analysis. *arXiv preprint arXiv:1701.08546* (2017).

[32] C. Sun, D. Lo, S. Khoo, and J. Jiang. 2011. Towards more accurate retrieval of duplicate bug reports. In *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. 253–262. https://doi.org/10.1109/ASE.2011.6100061

[33] Chengnian Sun, David Lo, Xiaoyin Wang, Jing Jiang, and Siau-Cheng Khoo. 2010. A discriminative model approach for accurate duplicate bug report retrieval. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 45–54.

[34] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun. 2008. An approach to detecting duplicate bug reports using natural language and execution information. In *2008 ACM/IEEE 30th International Conference on Software Engineering*. 461–470. https://doi.org/10.1145/1368088.1368151

[35] W Eric Wong, Ruizhi Gao, Yihao Li, Rui Abreu, and Franz Wotawa. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering* 42, 8 (2016), 707–740.

[36] Yang Wu, Ang Chen, Andreas Haeberlen, Wenchao Zhou, and Boon Thau Loo. 2017. Automated Bug Removal for Software-Defined Networks.. In *NSDI*. 719–733.

[37] Bowen Xu, Deheng Ye, Zhenchang Xing, Xin Xia, Guibin Chen, and Shanping Li. 2016. Predicting semantically linkable knowledge in developer online forums via convolutional neural network. In *Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering*. ACM, 51–62.

[38] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. 2016. Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports. In *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*. 127–137. https://doi.org/10.1109/ISSRE.2016.33

[39] Yiwen Yang, Jun Ai, and Fei Wang. 2018. Defect Prediction Based on the Characteristics of Multilayer Structure of Software Network. In *2018 IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*. IEEE, 27–34.

[40] Xin Ye, Razvan Bunescu, and Chang Liu. 2016. Mapping bug reports to relevant files: A ranking model, a fine-grained benchmark, and feature evaluation. *IEEE Transactions on Software Engineering* 42, 4 (2016), 379–402.

[41] Jian Zhou and Hongyu Zhang. 2012. Learning to Rank Duplicate Bug Reports. In *Proceedings of the 21st ACM International Conference on Information and Knowledge Management* (Maui, Hawaii, USA) *(CIKM'12)*. Association for Computing Machinery, New York, NY, USA, 852âĂŞ861.

[42] Thomas Zimmermann and Nachiappan Nagappan. 2007. Predicting Subsystem Failures Using Dependency Graph Complexities. In *Proceedings of the The 18th IEEE International Symposium on Software Reliability (ISSRE '07)*. IEEE Computer Society, Washington, DC, USA, 227–236. https://doi.org/10.1109/ISSRE.2007.19

[43] Thomas Zimmermann and Nachiappan Nagappan. 2008. Predicting defects using network analysis on dependency graphs. In *Proceedings of the 30th international conference on Software engineering*. ACM, 531–540.