How Incidental are the Incidents? Characterizing and Prioritizing Incidents for Large-Scale Online Service Systems

Junjie Chen[*]
College of Intelligence and Computing, Tianjin University
Tianjin, China
junjiechen@tju.edu.cn

Shu Zhang
Microsoft Research
Beijing, China
v-shuzh@microsoft.com

Xiaoting He
Microsoft Research
Beijing, China
v-xiah@microsoft.com

Qingwei Lin[†]
Microsoft Research
Beijing, China
qlin@microsoft.com

Hongyu Zhang
The University of Newcastle
Callaghan, Australia
Hongyu.Zhang@newcastle.edu.au

Dan Hao
Peking University
Beijing, China
haodan@pku.edu.cn

Yu Kang
Microsoft Research
Beijing, China
kay@microsoft.com

Feng Gao
Microsoft Azure
Redmond, USA
fgao@microsoft.com

Zhangwei Xu
Microsoft Azure
Redmond, USA
zhangxu@microsoft.com

Yingnong Dang
Microsoft Azure
Redmond, USA
yidang@microsoft.com

Dongmei Zhang
Microsoft Research
Beijing, China
dongmeiz@microsoft.com

ABSTRACT

Although tremendous efforts have been devoted to the quality assurance of online service systems, in reality, these systems still come across many incidents (i.e., unplanned interruptions and outages), which can decrease user satisfaction or cause economic loss. To better understand the characteristics of incidents and improve the incident management process, we perform the first large-scale empirical analysis of incidents collected from 18 real-world online service systems in Microsoft. Surprisingly, we find that although a large number of incidents could occur over a short period of time, many of them actually do not matter, i.e., engineers will not fix them with a high priority after manually identifying their root cause. We call these incidents *incidental incidents*. Our qualitative and quantitative analyses show that incidental incidents are significant in terms of both number and cost. Therefore, it is important to prioritize incidents by identifying incidental incidents in advance to optimize incident management efforts. In particular, we propose an approach, called **DeepIP** (**Deep** learning based **I**ncident **P**rioritization), to prioritizing incidents based on a large amount of historical incident data. More specifically, we design an attention-based Convolutional Neural Network (CNN) to learn a prediction model to identify incidental incidents. We then prioritize all incidents by ranking the predicted probabilities of incidents being incidental. We evaluate the performance of DeepIP using real-world incident data. The experimental results show that DeepIP effectively prioritizes incidents by identifying incidental incidents and significantly outperforms all the compared approaches. For example, the AUC of DeepIP achieves 0.808, while that of the best compared approach is only 0.624 on average.

[*]This work was mainly done when he was visiting Microsoft Research.
[†]Corresponding author.

ASE '20, September 21–25, 2020, Virtual Event, Australia
© 2020 Association for Computing Machinery.
ACM ISBN 978-1-4503-6768-4/20/09…$15.00
https://doi.org/10.1145/3324884.3416624

CCS CONCEPTS

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

KEYWORDS

Incidents, Online Service Systems, Prioritization

ACM Reference Format:
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 *35th IEEE/ACM International Conference on Automated Software Engineering (ASE '20), September 21–25, 2020, Virtual Event, Australia.* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/3324884.3416624

1 INTRODUCTION

In recent years, online service systems, such as Microsoft Azure and Office 365, have been widely used by millions of users around the world. To assure their quality, practitioners put dedicated efforts [10–12, 26, 29, 35, 36, 46, 53–55], but such online service systems still encounter many incidents (i.e., unplanned interruptions and outages). These incidents can decrease user satisfaction or cause serious economic loss. For example, the one-hour downtime for `Amazon.com` on Prime Day in 2018 (its biggest sale event of the year) caused the loss of up to $100 million [2]. Therefore, high availability and reliability are essential to online service systems.

Once an incident occurs to an online service system, it needs to be mitigated as soon as possible so as to reduce the loss caused by the incident [10, 29]. However, an online service system is quite complex, i.e., involving many components such as hardware, virtual machines, network, and database, and in the meanwhile all these components can lead to incidents in the daily operation of the system. Therefore, incidents tend to occur frequently in practice. Moreover, the number of engineers, who are responsible to deal with incidents, and computing resources is limited, and the cost spent on incident management is considerable. Therefore, it is scarcely possible to mitigate every incident timely. To reduce the impacts of incidents as much as possible, intuitively, one of the most cost-effective solutions is to deal with more important incidents earlier. That is, it is necessary to prioritize incidents for engineers to optimize the incident management process.

To achieve the goal of incident prioritization, we first need to understand what high-priority incidents and low-priority incidents are. However, to date, there lack extensive studies on incidents. Therefore, we perform the first empirical analysis for incidents of 18 real-world online service systems in Microsoft, including many worldwide popular systems. For each system, we collect incident data over a six-month period. Indeed, there are a large number of incidents reported over a short period. For example, for Service X there are nearly 3,000 incidents per time unit[1]. However, many of them actually are not important, i.e., they do not really affect customers and engineers will not *fix* them with a high priority *after* they manually *find the reasons* why the incidents occur. In this paper, we call these incidents *incidental incidents*. In contrast, we call the remaining incidents *essential incidents*.

More specifically, since essential incidents can be caused by a variety of factors (e.g., various source code bugs and hardware failures), it is difficult to characterize them thoroughly. Here, we aim to prioritize incidents from the opposite direction, i.e., identifying incidental incidents and then putting them in the end. Therefore, we conduct a qualitative analysis to characterize incidental incidents. We find that the incidents fall into several categories. Also, we conduct a quantitative analysis to investigate the impacts of incidental incidents. We find that for 15 out of 18 studied systems, the percentage of incidental incidents is more than 30% and the percentage of maintenance time spent on incidental incidents is also more than 30%! The results demonstrate that a large number of engineers' efforts were spent on incidental incidents, which can largely delay the mitigation of really important incidents. Those also empirically motivate the necessity of incident prioritization.

Further, in this paper we propose a deep-learning based approach, called **DeepIP** (**Deep** learning based **I**ncident **P**rioritization), to prioritizing incidents by identifying incidental incidents based on a large amount of historical incident data. In particular, there are two main challenges in the problem: 1) which features are helpful to identify incidental incidents; 2) how to effectively utilize these features to identify incidental incidents. For the first challenge, our empirical study provides some guidelines on effective features and helps us identify three types of features, i.e., textual descriptions (i.e., title and summary of an incident report), special terms (e.g., API names and component names occurring in an incident report), and incident-occurring environment information (e.g., incident-occurring device). To overcome the second challenge, we draw support from deep learning. Since the features we used are mainly textual information, deep learning can achieve semantic understanding of natural-language descriptions and outperform traditional machine learning algorithms, as demonstrated by existing studies [10, 16]. More specifically, we design a CNN-based deep neural network and incorporate an attention mechanism. The incidental incidents can be predicted by the attention-based CNN model. Then, we prioritize all the incidents based on the ascending order of the predicted probabilities of being incidental. In this way, engineers can optimize the incident management process by handling the incidents ranked higher first.

To investigate the effectiveness of our approach DeepIP, we conduct an extensive evaluation using real-world incident data from Microsoft (the same data as the one used in the empirical study). The results demonstrate that DeepIP is able to effectively and efficiently prioritize incidents for large-scale online service systems, and significantly outperform all the compared approaches in the area of software bug severity prediction[2]. For example, the average AUC (measuring the accuracy that essential incidents are ranked higher than incidental incidents) of DeepIP is 0.808, while that of the best compared approach is just 0.624. The results demonstrate that as the first attempt to solve the practical problem of incident prioritization, our deep-learning based approach DeepIP is indeed promising. Our study results also show that each type of features (i.e., special terms and incident-occurring environments) can significantly improve the effectiveness of DeepIP, confirming the contributions of each of them. In particular, the practical value of DeepIP has been appreciated by engineers in Microsoft.

The major contributions of the paper are as follows:

- We perform the first large-scale empirical study on incidents of 18 real-world online service systems, characterizing incidents qualitatively and quantitatively.
- We propose the first approach to prioritizing incidents by identifying incidental incidents, in order to optimize the incident management process.
- We conduct an extensive study to evaluate the performance of DeepIP based on real-world incident data of 18 large-scale online service systems. Our results show that DeepIP significantly outperforms all the compared approaches.

2 BACKGROUND

In this section, we introduce the background of incidents and incident management (IcM) for online service systems in practice.

[1]Due to the company policy, we hide the time unit and service name.

[2]Since there is no existing incident prioritization approach for online service systems, we adapt the typical approaches of traditional software bug severity prediction for comparison in our study.

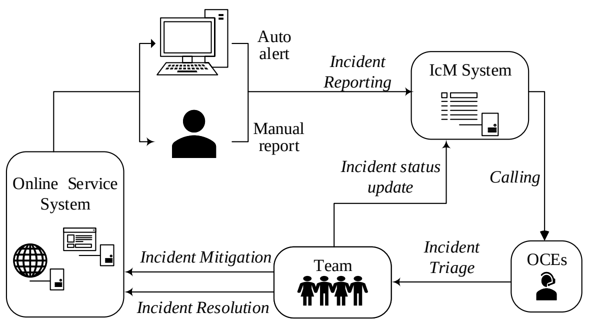


Figure 1: Workflow of Incident Management

Figure 1 shows the workflow of IcM for large-scale online service systems in Microsoft. The IcM consists of four stages. The first stage is *incident reporting*. In an online service system, a large number of monitors are used to watch for some key performance indicators (e.g., latency and network status). Most incidents are actually reported by various monitors automatically, which is different from the manual reporting of traditional software bugs. In particular, a monitor automatically reports an incident to the IcM system when some predefined anomalous conditions are met. Besides, engineers could observe incidents during their daily operation, and thus they could also report incidents manually to the IcM system. After an incident is reported to the IcM system, the IcM system first makes a phone call to a set of On-Call Engineers (OCEs) to trigger the investigation process of the incident. Ideally, the OCEs can directly identify the root cause and then mitigate the incident based on the information in the incident report. However, in most cases they cannot find the root cause within a short time. Therefore, the OCEs have to assign the incident to a team that they think is the most suitable to handle it, which is the second stage *incident triage*. The third stage is *incident mitigation*. When the incident is assigned to the correct team, the engineers in the team begin to diagnose the problem and then take all necessary actions to mitigate the incident (e.g., reboot the server or replace failure hardware). After mitigation, the engineers further analyze the underlying root cause of the incident through offline postmortem analysis, and finally completely resolve the incident, which is the last stage *incident resolution*. Moreover, the engineers update the status (e.g., mitigated or resolved) of an incident in the IcM system.

Incident mitigation should be timely since long Time To Mitigate (TTM) could lead to poor service availability and cause huge financial loss. However, it is often costly for engineers to manually mitigate an incident due to the complexity and scale of the system. Considering the large number of incidents as well as the limited number of engineers and computing resources, it is essential to handle more important incidents earlier. However, to our best knowledge, none of existing work has studied the priority and influence of incidents before. Therefore, in this paper, we conduct the first extensive study to characterize incidents (presented in Section 3), and further propose an effective approach to prioritizing incidents to optimize the incident management process (presented in Section 4).

3 AN EMPIRICAL STUDY OF INCIDENTS

To better understand the priority and influence of incidents for large-scale online service systems, we conduct the first extensive study on real-world incidents. As presented in Section 1, in this study, we aim to understand the characteristics of incidental incidents. Based on the identification of the incidental incidents, we can prioritize incidents by putting the incidental incidents in the end. Here, we target at the following research questions:

- **RQ1**: Which incidents are incidental in large-scale online service systems?
- **RQ2**: What is the percentage of incidental incidents?
- **RQ3**: What is the cost spent on incidental incidents?
- **RQ4**: Is the current incident management practice good enough?

RQ1 is qualitative analysis to investigate "what are incidental incidents", while the other RQs are quantitative analysis to study incidental incidents in terms of number and cost. In particular, RQ4 aims to explore whether the current practice of incident management is sufficiently good from the view of incident priority.

3.1 Subjects

We used 18 large-scale online service systems in Microsoft as subjects. These subjects include many worldwide popular products and are used by millions of users worldwide. All the 18 systems are in different application areas and developed by different product groups, indicating the diversity of subjects. For each online service system, we collected real incident data over a six-month period. The size of all the collected incident reports is up to 4.2GB. The total number of monitors used to monitor these systems is more than 80K. Due to the policy of Microsoft, we hid some details such as the specific time period for incident collection and the specific number of collected incidents.

3.2 RQ1: Qualitative Analysis

After diagnosing an incident, engineers tend to manually record whether the incident needs to be fixed with a high priority, and give a simple explanation for the incident in the IcM system. If the incident indeed needs to be fixed with a high priority (i.e., it is an essential incident), engineers also record the fixing steps. That also means, although an incident is an incidental incident, engineers still have to spend time and resources on diagnosing the reason why it occurs, and finally find that it actually is an incidental one. With the help (i.e., manual recording) of engineers, incidental incidents can be divided into six categories: *by design*, *customer error*, *won't fix*, *unable to reproduce*, *transient*, and *false alarm*. We analyze each category of incidental incidents in the following.

3.2.1 By Design. The incidents belonging to this category are produced intentionally, and do not need to be dealt with. This category of incidents tends to have the purpose of testing. One of such incidents is shown in Example 1. According to the title (one line description) of the incident report and the explanation given by the engineers who diagnosed the incident, obviously, this incident is an intentional incident for testing *OData API*. The occurrences of this category of incidents are as expected, and thus they should be assigned low priority in practice.

Example 1:

Incident: *TestIncident2018-10-23T00:06:36Z.*
Explanation: *OData API Test.*

3.2.2 Customer Error. This category of incidents is caused by customer errors rather than the problems of online service systems. In other words, customers misuse the systems or incorrectly configure the systems, causing the occurrences of the incidents. For example, as shown in Example 2, the incident is that sending email notification failed. However, the cause for this incident was that the inbox of the customer was full. Similarly, in Example 3, the cause for the incident "badge preview missing at DSM" was that the DYMO driver was not correctly installed by customers. This category of incidents can be directly handled by the technical supporters, and thus do not need to notify engineers.

Example 2:

Incident: *Failed to send email notification on Tenant: 022d9fca-60a3-4aac-9a90-c18e51ac527e at Frequency: Daily.*
Explanation: *Inbox is full - insufficient storage error.*

Example 3:

Incident: *Badge Preview Missing at DSM.*
Explanation: *Customer installed DYMO driver and resolved the issue.*

3.2.3 Won't Fix. The incidents belonging to this category are real incidents, but they tend to occur at the parts that are out-of-date (i.e., not maintained anymore). Therefore, it is not worth taking the engineers' efforts to solve them. As shown in Example 4, engineers think that it is not necessary to solve the incident "node service stuck in crash loop", since the occurring part of the incident has been deprecated.

Example 4:

Incident: *Node service stuck in crash loop on BR1-NNS207.*
Explanation: *Deprecated fabric.*

3.2.4 Unable to Reproduce. The incidents belonging to this category are not able to be reproduced during diagnosis. In this case, it is hard to say whether they are real incidents, and it is scarcely possible to check whether we can solve them successfully. Therefore, even if the incidents are reported, they have to be ignored.

3.2.5 Transient. This category of incidents is real things but they can be automatically recovered. These incidents tend to be caused by other operations/factors. When these operations/factors are corrected/eliminated, the incidents can be automatically resolved. Therefore, it should be low-priority for engineers to look into these incidents. For example, as shown in Example 5, the test `EndToEndDataPushAndPull` was not executed during a period of time. Actually, it was caused by another factor, which is that the corresponding machines have not been completely rebooted at that time. When the machines finished rebooting, the test would run. Similarly, in Example 6, the incident occurred, since the "msf Forest" was still at the stage of deployment. After the deployment finished, the incident was automatically resolved accordingly.

Example 5:

Incident: *WestCentralUS: GIP2 test EndToEndDataPushAndPull didn't execute at least once during the last 14:00:00 minutes.*
Explanation: *The argo was down because the machines took long time to reboot (ntdev texas password was changed). Argo is up now and tests are running.*

Example 6:

Incident: *RED ALERT: System Level Issue Detected in msf Forest.*
Explanation: *Transient issue due to deployments. There are no users in this forest and server is not member of any DAG.*

3.2.6 False Alarm. The incidents belonging to this category are actually not real incidents. They tend to be caused by the problems of monitors. For example, as shown in Example 7[3], this incident is a false alarm, and the real cause lay in the monitor reporting the incident. More specifically, the data was updated every three minutes but the monitor checked every one minute, and thus the monitor reported the incident "refresh time exceeded threshold". However, the incident was actually due to the sensitive monitor (improper threshold). That is, this category of incidents should not be investigated by engineers, since they are not real incidents.

Example 7:

Incident: *DS002 (MDM): Refresh time of delta store "StorageAccountName":"xtlcsuse", "**TableName":"DeltaStore", "Name":"envindex" exceeded threshold.*
Explanation: *MDM was configured for this at a threshold of 1 minute but the new code is at 3 minutes. We did not get enough 3 minute outages to trigger this, so this is a false alarm.*

3.3 RQ2: Percentage of Incidental Incidents

We investigated the percentage of incidental incidents in online service systems, whose results are shown in Figure 2. In this figure, the value above each bar represents the percentage of incidental incidents among all the incidents for the corresponding subject. From this figure, we find that the percentage of incidental incidents for all the 18 studied systems is significant, which ranges from 11.92% to 71.43%. The average percentage of them is up to 50.32%. That is, more than half of incidents are actually incidental, indicating that a great deal of engineers' efforts were spent on these low-priority incidents during historical diagnosis. Therefore, it is quite necessary to understand and then prioritize incidents for online service systems. In particular, we investigated why *S9* has the smallest rate, and found that the number of monitors used for checking *S9* is the smallest, which may lead to many incidents that cannot be reported and hard to capture complex interactions among components.

[3]We use ∗∗ to replace some words due to the company policy.

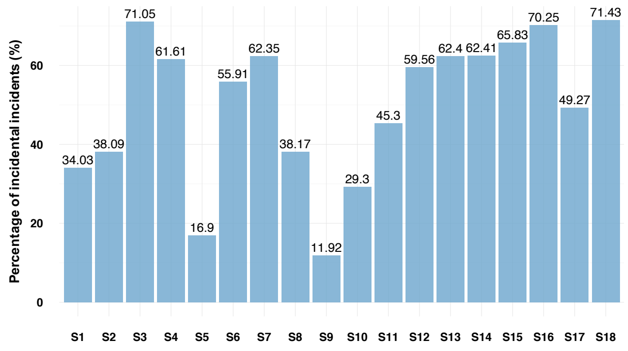


Figure 2: Percentage of incidental incidents

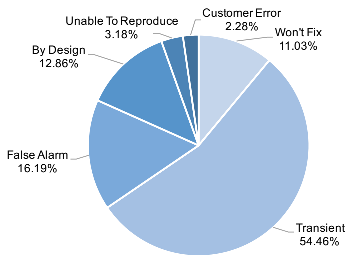


Figure 3: Percentage of each category of incidental incidents

We further investigated the percentage of each category of incidental incidents, whose results are shown in Figure 3. The numbers in this figure represent the average percentage of the corresponding category of incidental incidents on the 18 studied systems. In this figure, we find that the percentages of "Customer Error" and "Unable to Reproduce" are small, while the percentage of "Transient" is large. The large percentage (i.e., 54.46%) for "Transient" is as expected, since there exist many interactions among various components in a large-scale online service system. When one component is abnormal, the components interacting with it are likely to report incidents that are caused by the first abnormal one, which may lead to many "Transient" incidents.

3.4 RQ3: Effort Spent on Incidental Incidents

We explored the effort spent on incidental incidents in terms of TTR (Time to Resolve). TTR refers to the time period from incident creation to incident resolution. Figure 4 shows the percentage of TTR spent on incidental incidents for each online service system. From this figure, we can see that the percentage of TTR spent on incidental incidents is significant, ranging from 10.57% to 76.72%. The average percentage is up to 55.05%. That is, the cost spent on incidental incidents is almost the same as that spent on essential incidents in terms of TTR, which may delay the resolution of essential incidents and thus result in greater economic loss. We also find that for 15 out of 18 studied systems, the percentage of resolution time spent on incidental incidents is more than 30%. These results further motivate the necessity of understanding and prioritizing incidents for large-scale online service systems.

We further explored the TTR spent on each category of incidental incidents. We first calculated the average TTR of each category of incidental incidents for each studied system, and then showed the average TTR distribution across all the subjects in Figure 5. Due to the policy of Microsoft, we hid the specific time and its unit. In Figure 5, the violin plots show the density of average TTR at different values, and the box plots show the median and interquartile ranges. Among all these categories of incidental incidents, the median of average TTR for "Unable to reproduce" is the largest. The reason could be that developers spent long time trying to reproduce these incidents but failed.

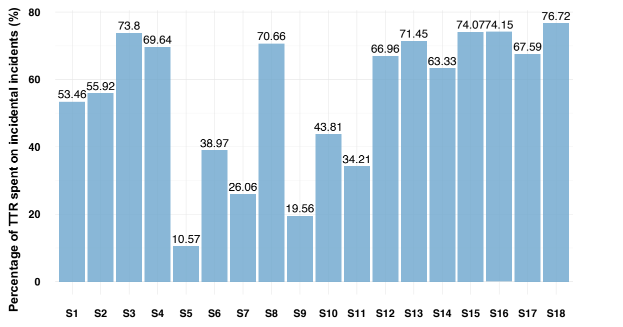


Figure 4: Percentage of TTR spent on incidental incidents

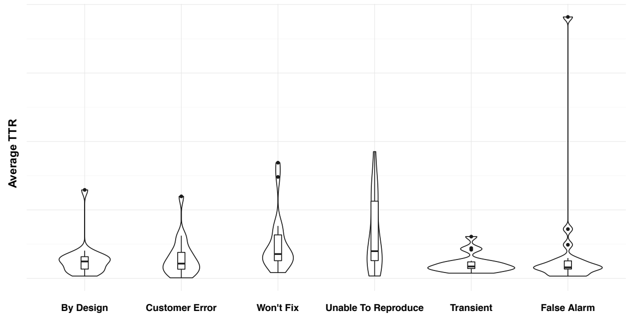


Figure 5: Distribution of TTR

3.5 RQ4: Investigation of Incident Management Practice

The current incident management practice in Microsoft is that engineers investigate the reported incidents based on the number of potentially impacted customers, which is estimated according to the region/cluster where the incidents occurred. Based on the number of potentially impacted customers, there are five levels of incidents in Microsoft, i.e., $0 \sim 4$, where 0 refers to the highest level and 4 refers to the lowest levels. However, in the current practice engineers may still investigate some incidental incidents first since these incidents have larger estimated impacts, which is harmful to incident management of systems. Therefore, we explored the distribution of incidents at each level to investigate whether the current practice is good enough.

Table 1 shows the distribution of incidents at each level. In this table, Rows 2-7 present the distribution of each category of incidental incidents at each level across all the studied systems. Rows 8 and 9 summarize the overall distribution of incidental incidents and essential incidents at each level. From this table, we can see

Table 1: Distribution of incidents at each level

Severity	0 (%)	1 (%)	2 (%)	3 (%)	4 (%)
By Design	13.52	5.06	3.54	4.70	6.77
Customer Error	0.00	0.45	0.41	0.67	1.73
Won't Fix	2.96	6.43	6.74	3.20	6.41
Unable To Reproduce	0.00	0.83	2.28	0.64	2.47
Transient	30.37	9.25	23.04	21.62	21.36
False Alarm	11.11	3.75	6.22	5.55	8.84
Incidental Incidents	57.96	25.77	42.23	36.38	45.57
Essential Incidents	42.04	74.23	57.77	63.62	52.43

that at each level, there are both incidental incidents and essential incidents, and their rates are relatively similar. That is, there are indeed many incidental incidents that are assigned with a higher level than actual essential incidents. Even at the highest level, the percentage of incidental incidents (i.e., 57.96%) is larger than that of essential incidents (i.e., 42.04%). That indicates, many incidental incidents were actually investigated preferentially, causing the waste of engineers' efforts. Therefore, the current incident management practice should be further improved, and one promising direction is to prioritize incidents by identifying essential incidents and incidental incidents.

4 INCIDENT PRIORITIZATION

During the incident management process, a huge amount of labeled incident data is accumulated. Each incident is reported along with various information such as the symptom description and occurring environment. The abundance of data provides an opportunity to automatically identify whether an incident is incidental or essential. Here, we treat the problem of identifying incidental incidents as a supervised classification problem, which can produce a probability of an incident being incidental. Then, all incidents can be prioritized based on the ascending order of the predicted probabilities. In this way, engineers can handle the incidents based on the priorities and the incident management process could be improved.

Here, we propose a deep-learning based approach, called **DeepIP**, to prioritizing incidents by identifying incidental incidents. Figure 6 shows the overview of DeepIP. We identify three types of features to help predict incidental incidents based on the guidelines acquired from the empirical study (Section 4.1). Also, we design a CNN-based deep neural network and incorporate an attention mechanism to effectively utilize these features for the identification of incidental incidents (Section 4.2). Here, we draw support from deep learning, since the features we used are mainly textual information, and deep learning can achieve semantic understanding of natural-language descriptions and has been demonstrated to outperform traditional machine learning algorithms [10, 16].

4.1 Feature Identification

When an incident is reported, it is provided with the textual description about the symptom, i.e., the title and summary of the incident report. The textual description is the core information about an incident, which can directly reflect the incident to some degree, and thus it could be helpful to distinguish whether the incident is essential or incidental. Besides, from Section 3 and the existing study [10], many incidents are actually correlated in a system. For example, an incident in one component of a system could cause a series of incidents in other components of the system. Also, an incident may be continuously reported several times, since monitors check the status of a system regularly. "Transient" incidents are also related to incident correlations. For ease of presentation, we call the incident to be predicted *target incident* and the incidents correlated with it *relevant incidents*. The relevant incidents are helpful to predict the target incident. Therefore, we consider the textual descriptions of both the target incident and its relevant incidents in DeepIP. As the relevant incidents tend to be reported at close time [10] with the target incident, we identify them by setting a time window and collecting the incidents whose reporting time is within the window and before the time of the target incident. In summary, *the first type of features used in DeepIP is the textual descriptions in both the target incident report and its relevant incident reports.*

Based on the observations in Section 3, most of incident reports include special terms, such as API names and component names, and many of them are helpful to identify incidental incidents. For example, as shown in Example 4, "BR1-NNS207" was deprecated, and thus this incident did not need to resolve, belonging to "Won't Fix". Here, "BR1-NNS207" is a special term. As shown in Example 6, "msf Forest" is also a special term. This incident is "Transient", which occurred since "msf Forest" was still at the stage of deployment. Therefore, *the second type of features used in DeepIP is special terms in the target incident report.* Actually, special terms have been included in textual descriptions in incident reports (the first type of features). However, during the learning of textual descriptions, the knowledge of special terms is hard to learn since the frequencies of special terms are much smaller than those of other words in textual descriptions. Therefore, to effectively learn the knowledge of special terms, we extract special terms as a kind of features based on their frequencies.

According to Section 3, the incident-occurring environments are also related to the identification of incidental incidents. For example, the incidents belonging to "False Alarm" tend to be caused by the problems of monitors, and thus the monitor ID reporting an incident is helpful to distinguish whether the incident is essential or incidental. Therefore, *the third type of features used in DeepIP is the incident-occurring environmental information.* In particular, we consider the following environmental information: monitor ID, incident-occurring device, and incident-reporting type.

4.2 Design of Deep Neural Network

To effectively utilize the three types of features to identify incidental incidents, we design a deep neural network for DeepIP. In the following, we first present feature embedding in Section 4.2.1, then introduce attention-based text encoding in Section 4.2.2, and finally present incidental incident prediction in Section 4.2.3.

4.2.1 Feature Embedding. Since the values of the second and third types of features used in DeepIP are a finite set of discrete values, we conduct feature embedding for them. One simple method is to express all these discrete values as one-hot vectors [25]. However, in this way, the dimension of an one-hot vector may be very high, and it ignores the possible relations among different feature values. To overcome these problems, we adopt representation learning [7] to embed each feature value into a vector. Representation learning is able to embed a value to a fixed-dimension vector, and gradually

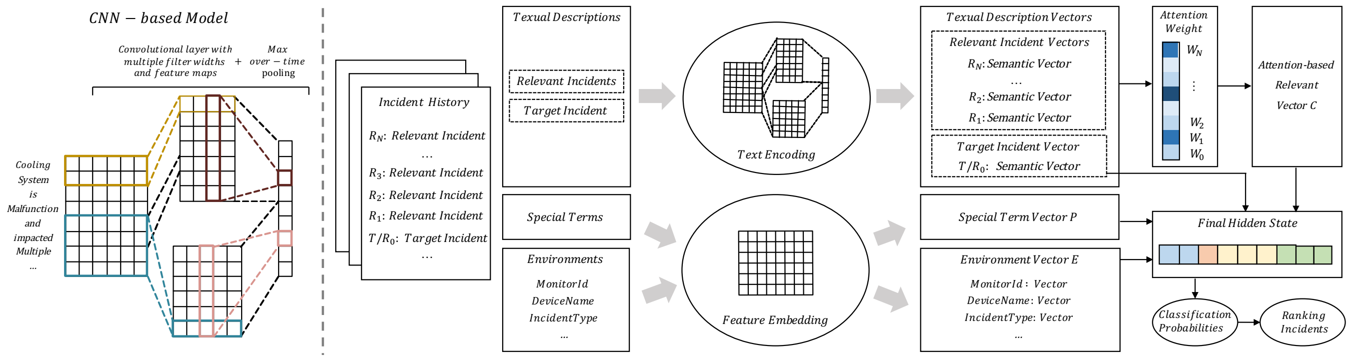


Figure 6: Overview of DeepIP

updates the vector during the training process. In this way, each of these features is embedded into a fixed-dimension vector. More specifically, we denote the j_{th} feature vector in the second types of features as $P_j = \{p_{j1}, p_{j2}, \ldots, p_{jn}\}$, and denote the k_{th} feature vector in the third types of features as $E_k = \{e_{k1}, e_{k2}, \ldots, e_{ks}\}$, where n and s refer to the pre-defined fixed dimensions in representation learning for the second and third types of features, respectively. Here, we concatenate all the second types of feature vectors into a vector $P = P_1 \oplus P_2 \oplus \ldots \oplus P_t$, and concatenate all the third types of feature vectors into a vector $E = E_1 \oplus E_2 \oplus \ldots \oplus E_r$, where t is the number of the second type of feature vectors, r is the number of the third type of feature vectors, and $\oplus$ is the concatenation operator.

4.2.2 *Attention-based Text Encoding.*

After acquiring the target incident and its relevant incidents, for each of these incidents, DeepIP first applies standard text mining method [8] to process the textual description (including tokenization, removing stop words, and splitting). DeepIP then embeds each word in the textual description of an incident report into a vector by using a word vector, which is pre-trained on historical incident data using the FastText algorithm [28]. In this way, the textual description of an incident is transformed into a matrix, in which the number of rows is equal to the number of words in the textual description.

Then, DeepIP encodes the matrix for an incident into a vector. We use a CNN based neural-language model rather than traditional language models to conduct encoding, since the former has been demonstrated to be able to encode more complex patterns and focus on word-level knowledge to achieve better performance [27, 31]. In particular, we adopt the simple single-layer CNN-based model [31] for encoding, since it has been demonstrated that such a model can achieve better or comparable results and is easy for training and prediction [31]. An overview of the CNN-based model is shown in the left figure of Figure 6. More specifically, the CNN-based model contains multiple 1D (1-dimension) convolution kernels and max-over-time pooling. By using multiple convolution kernels with several different widths, several feature maps are produced from the matrix, where multiple convolution kernels with different widths can capture the correlation among different numbers of adjacent words. Then, the max-over-time pooling is applied to produce a vector for each feature map in order to extract the most important words. Finally, all produced vectors are concatenated to generate a vector for an incident. In this way, the textual description of the target incident or each of the relevant incidents is encoded into a vector. We denote the vector of the target incident as $T = \{t_1, t_2, \ldots, t_m\}$, and the vector of the i_{th} relevant incident as $R_i = \{r_{i1}, r_{i2}, \ldots, r_{im}\}$, where m is the total number of convolution kernels.

Since different relevant incidents may have different degrees of correlation with the target incident, it is necessary for DeepIP to learn these different degrees. The relevant incident that has a stronger correlation with the target incident should be assigned with a larger weight. Here, we introduce an attention mechanism to automatically learn the weights of relevant incidents. Actually, it is possible that all the identified relevant incidents have no correlation with the target incident, and thus we also consider the correlation of the target incident with itself when learning weights, so as to avoid assigning weights to the irrelevant incidents in this case. For ease of description, we call the vector of the target incident *the* 0_{th} *relevant incident* and denote T as $R_0 = \{r_{01}, r_{02}, \ldots, r_{0m}\}$. The input to the attention mechanism is the vectors of the target incident and its relevant incidents. The output is the attention-based relevant vector integrated by these vectors, denoted as $C = \{c_1, c_2, \ldots, c_m\}$, where $c_i = \sum_{j=0}^{N}(w_j * r_{ji})$ and w_j is the learned weight of the j_{th} relevant incident. The calculation of w_j is shown as Formula 1.

$$w_j = \frac{e^{f(R_0, R_j))}}{\sum_{k=0}^{N} e^{f(R_0, R_k)}} \tag{1}$$

where, $f(A, B) = v^{\top} tanh(W_A A + W_B B)$, A and B are two vectors, and v, W_A, and W_B are parameters learned in the MLP [6].

4.2.3 *Predicting Incidental Incidents.*

After acquiring the four vectors (i.e., T, C, P, and E), DeepIP first utilizes the four vectors to construct the final hidden state. Here, DeepIP concatenates the four produced vectors T, C, P, and E, i.e., $T \oplus C \oplus P \oplus E$. Then, DeepIP converts the final hidden state into a probability distribution of labels by the last layer, i.e., the softmax layer. In particular, DeepIP minimizes the loss by gradient descent and gradually updates the weight of the network. The classification cross entropy loss is used for training. In this way, the incidental incidents can be predicted, and each incident is assigned with a probability of being incidental.

At last, all the incidents are prioritized based on the ascending order of the predicted probabilities. The higher the incidents are ranked, the larger the probabilities that the incidents are essential incidents are. Engineers can then optimize their incident management process by handling the incidents ranked higher first. Moreover, since our attention mechanism is able to learn the correlation degrees of relevant incidents with the target incident, DeepIP also recommends the most relevant incident together with the target incident, which could facilitate engineers to understand the nature of the incident and diagnose it.

5 EVALUATION

To investigate the performance of DeepIP, we conduct an extensive study using real-world incident data. We use the 18 industrial online service systems studied in Section 3 as subjects. In particular, we use the incident data from the first four months as the training data, and the incident data from the last two months as the testing data. In the study, we address the following research questions:

- **RQ5:** How does DeepIP perform for real-world large-scale online service systems?
- **RQ6:** Does each type of features contribute to DeepIP?

5.1 Evaluation Design

5.1.1 Compared Approaches. Since our work is the first to prioritize incidents by predicting incidental incidents for online service systems, there is no direct comparative approach. For traditional software systems, there are several work on predicting severity of bug reports, which can be adapted to prioritize incidents [32, 37, 43, 52]. Therefore, we select two typical bug-severity prediction approaches as the comparative approaches in this study. Both approaches are also based on textual descriptions of reports.

- Menzies and Marcus [37], which applies the standard text mining method [8] to process textual descriptions in reports, and then uses tf-idf (term frequency and inverse document frequency) [24] to transform the textual descriptions in a bug report to a vector. Finally, their approach uses the rule classifier based on entropy and information gain to predict bug severity [39].
- Lamkanfi et al. [32], which applies the standard text mining method to process textual descriptions. Then, this approach counts token frequency and uses the Naive Bayes algorithm to predict bug severity.

When adapting these approaches, the incident severity is incidental and essential, and all the incidents are prioritized based on the ascendant order of the predicted probabilities that incidents are incidental, which are given by the corresponding approach. For ease of presentation, in this paper we call the two approaches *Rule* and *Bayes* respectively, based on the way they perform prediction.

Actually, we also tried to use another state-of-the-art approach to bug severity prediction [52] as a comparative approach. This approach calculates the similarities between a new bug report and historical bug reports using BM25F [41] and LDA [9]. However, it cannot work well on incidental incident prediction since it is very time costly. Its time complexity is $O(mn)$, where m and n are the number of incidents in training and testing data, respectively. For each instance in testing data, it has to calculate the similarities with all incidents in training data, thus its cost is very considerable due to the large scale of the incident data for a system in practice. For example, we applied it to the system with the smallest number of incidents in our study (i.e., S7). The time spent on prioritizing incidents in testing data is up to 3,330 seconds, while the time required by DeepIP, *Rule*, and *Bayes* is only 2.54 seconds, 0.01 seconds, and 0.31 seconds as shown in Table 2. For larger datasets, the required time could be much longer. Therefore, we did not include this approach as a comparative approach in our study.

In RQ6, we evaluate the contribution of each type of features. Here, we always keep the first type of features (i.e., textual descriptions) since it is the core information about incidents. We remove the second or third type of features and produce two variants of DeepIP, denoted as DeepIP$_{noP}$ and DeepIP$_{noE}$, respectively. We then compare the performance of the three versions.

5.1.2 Implementations and Parameters. Since the implementations of compared approaches are unavailable, we re-implemented them following descriptions in the papers. For the involved machine learning algorithms, we adopted the implementations provided by scikit-learn [3]. For DeepIP, we implemented CNN based on Apache MXNet [1], a scalable deep learning framework. For the compared approaches, we used the same values of the parameters as given in the corresponding papers. If a parameter's value is not explicitly given in the paper, we used the default value provided by the adopted tools. For the parameters in DeepIP, we determined them through grid search and set the same parameters for all the studied systems. More specifically, we set the parameters as follows: the CNN uses three sets of convolution kernels with different widths (i.e., 3, 4, 5), each of which has 100 kernels, and the used epoch is 20. The time window is set to be the time interval backwards 10 incidents from the target incident. In Section 6, we discuss the impact of the time window on DeepIP. Our study is conducted on Windows Server 2016 with 28-core Dual-Intel Xeon E5-2690 CPU(2.60GHz), 512 GB memory, 64-bit operating system, and a single NVIDIA Tesla K80 GPU accelerator. We cannot release the incident data used in our study due to the policy of Microsoft, but we release the source code for these approaches in the project homepage: https://github.com/JunjieChen/DeepIP.

5.1.3 Metrics. In this study, we consider both effectiveness and efficiency to measure the performance of DeepIP. We first measure the effectiveness of DeepIP in incident prioritization by adopting the widely-used *AUC* metric [18], which measures the accuracy that essential incidents are ranked higher than incidental incidents in our context. That is, AUC can be viewed as a metric based on pairwise comparisons between classifications of the two classes. Following existing work [13, 14], supposing the output probabilities of an approach on the essential incidents are $\{x_1, x_2, \ldots, x_m\}$ and the output probabilities on the incidental incidents are $\{y_1, y_2, \ldots, y_n\}$, the AUC is computed as Formula 2. Larger is better.

$$AUC = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} 1_{x_i > y_j}}{mn} \tag{2}$$

Besides, since the identification of incidental incidents is the core of DeepIP, we also measure the effectiveness of the classification of incidental and essential incidents. Here, we adopted the widely-used *Precision* and *Recall* metrics. *Precision* is computed by $\frac{TP}{TP+FP}$, while *Recall* is computed by $\frac{TP}{TP+FN}$, where TP, FP, and FN refer to the number of true positives (TP), false positives (FP), and false

Table 2: Performance comparison among the *DeepIP*, *Rule*, and *Bayes* approaches

Sub	AUC			Precision			Recall			Training Time (s)			Predicting Time (s)		
	DeepIP	Rule	Bayes	DeepIP	Rule	Bayes	DeepIP	Rule	Bayes	DeepIP	Rule	Bayes	DeepIP	Rule	Bayes
S1	**0.915**	0.698	0.737	**0.808**	0.705	0.738	**0.800**	0.698	0.724	5586.71	1681.51	**62.76**	55.48	**1.82**	19.05
S2	**0.765**	0.605	0.510	**0.687**	0.601	0.570	**0.679**	0.601	0.510	16743.19	**97.12**	756.16	130.51	**0.12**	468.29
S3	**0.879**	0.688	0.439	**0.794**	0.693	0.445	**0.681**	0.688	0.438	2482.99	42.80	**30.59**	14.92	**0.05**	12.84
S4	**0.824**	0.675	0.449	**0.721**	0.686	0.408	**0.745**	0.675	0.449	2254.65	29.28	**26.19**	17.04	**0.08**	27.37
S5	**0.880**	0.621	0.691	**0.642**	0.535	0.549	**0.778**	0.621	0.691	5482.05	4.77	**3.20**	122.05	**0.10**	8.93
S6	**0.848**	0.669	0.652	**0.792**	0.675	0.657	**0.787**	0.669	0.653	2597.75	9.78	**6.06**	7.86	**0.02**	2.82
S7	**0.721**	0.584	0.516	**0.700**	0.584	0.539	**0.693**	0.584	0.515	657.72	3.37	**0.84**	2.54	**0.01**	0.31
S8	**0.809**	0.630	0.577	**0.755**	0.629	0.588	**0.756**	0.630	0.565	3150.25	**62.68**	122.30	7.07	**0.05**	35.00
S9	**0.824**	0.537	0.522	**0.656**	0.524	0.510	**0.666**	0.537	0.521	1509.49	9.87	**4.77**	6.92	**0.01**	3.37
S10	**0.925**	0.750	0.763	**0.848**	0.753	0.719	**0.846**	0.750	0.562	68792.16	489.68	**9.21**	139.52	**0.35**	4.30
S11	**0.686**	0.581	0.536	**0.657**	0.582	0.580	**0.654**	0.581	0.534	1563.38	48.01	**35.37**	7.98	**0.05**	27.17
S12	**0.783**	0.616	0.584	**0.773**	0.645	0.583	**0.747**	0.616	0.579	5518.76	**118.45**	149.06	24.74	**0.14**	78.69
S13	**0.850**	0.619	0.616	**0.754**	0.616	0.673	**0.772**	0.617	0.609	14627.12	**258.57**	269.80	87.92	**0.29**	159.02
S14	**0.751**	0.588	0.671	**0.680**	0.591	0.620	**0.673**	0.588	0.616	4839.38	140.64	**2.54**	29.53	**0.11**	1.64
S15	**0.704**	0.570	0.496	**0.651**	0.579	0.494	**0.633**	0.570	0.497	1402.75	28.11	**13.25**	12.62	**0.04**	10.36
S16	**0.804**	0.602	0.530	**0.691**	0.583	0.543	**0.708**	0.602	0.529	982.64	**20.01**	23.94	2.18	**0.02**	6.15
S17	**0.790**	0.660	0.710	**0.741**	0.662	0.740	**0.740**	0.662	0.710	9709.73	**61.36**	71.97	54.30	**0.04**	29.77
S18	**0.792**	0.534	0.546	**0.715**	0.534	0.554	**0.714**	0.534	0.546	1116.51	11.51	**7.35**	1.35	**0.01**	1.04
Avg	**0.808**	0.624	0.586	**0.726**	0.621	0.584	**0.726**	0.624	0.569	8278.74	173.20	**88.63**	40.25	**0.18**	49.78

negatives (FN), respectively. In particular, as demonstrated in the existing work [19], we calculate the mean of all the classes for the two metrics. Larger is better.

For efficiency, we record the time spent on the training process and the time spent on prioritizing all the incidents in testing data. We call the former *training time* and the latter *predicting time*.

5.2 Results and Analysis

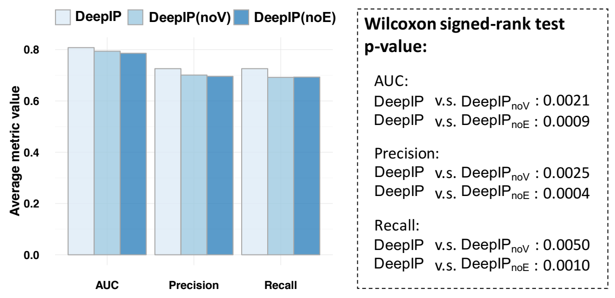
5.2.1 Overall Effectiveness of DeepIP. Table 2 shows the performance comparison among the three approaches, i.e., *DeepIP*, *Rule*, and *Bayes*, where bold values represent the best results among the three approaches. From Columns 2-4 in Table 2, DeepIP outperforms the two compared approaches (i.e., *Rule* and *Bayes*) on all the studied systems in terms of AUC, showing the significant advantage of DeepIP for incident prioritization. In particular, the average AUC of DeepIP is 0.808, while the other two approaches are just 0.624 and 0.586, respectively. Also, the range of AUC for DeepIP is from 0.686 to 0.925, demonstrating that DeepIP is able to achieve stably good prioritization effectiveness. Furthermore, in terms of *Precision* and *Recall* shown in Columns 5-10 in Table 2, DeepIP also performs much better than *Rule* and *Bayes* on all the subjects. The average *Precision* and *Recall* of DeepIP are 0.726 and 0.726, while those of *Rule* are 0.621 and 0.624 and those of *Bayes* are 0.584 and 0.569. These results also reflect that the adapted approaches from traditional bug-severity prediction are not suitable to solve the problem of incidents of online service systems due to their differences. We further analyze the reason is that, incidents are mainly automatically reported by monitors and have significant time and location correlations [10], while bug reports are manually reported by users and thus the correlations are not as significant as in incident reports. Therefore, existing approaches for bug reports ignoring correlations, cannot perform well for incidents. That is, it is quite necessary to propose new approaches by considering the characteristics of incidents. For incident prioritization, DeepIP is the first attempt to solve this problem, and our results have shown that it is a promising approach.

Columns 11-16 in Table 2 show the efficiency comparison among *DeepIP*, *Rule*, and *Bayes*. For the *offline* training time, *Bayes* spends the shortest time (i.e., 0.02 hours), while our approach DeepIP spends the longest time (i.e., 2.30 hours) on average. However, the training is offline and thus several hours are acceptable in practice. For the *online* predicting time, *Rule* is the most efficient one (i.e., 0.18 seconds) while *Bayes* spends the longest time (i.e., 49.78 seconds) and DeepIP is the medium (i.e., 40.25 seconds) on average. Actually, the online predicting time of all the three approaches is short, and even the time spent on predicting one incident is negligible. Therefore, our approach DeepIP is practical due to the short online predicting time and acceptable offline training time.

5.2.2 Contributions of Different Types of Features. Figure 7 shows the comparison among DeepIP, DeepIP$_{noP}$, and DeepIP$_{noE}$, where the Y-axis represents the average metric value of the 18 subjects. From the left figure, DeepIP performs better than both DeepIP$_{noP}$ and DeepIP$_{noE}$ in all the metrics. We further conducted a Wilcoxon signed-rank test [47] at the significant level of 0.05 between DeepIP and DeepIP$_{noP}$/DeepIP$_{noE}$ to investigate whether the former significantly outperforms the latter two. From the right of Figure 7, we find that all the p-values are much smaller than 0.05, demonstrating that DeepIP indeed significantly outperforms DeepIP$_{noP}$ and DeepIP$_{noE}$ in terms of AUC, Precision, and Recall. The results indicate that both special term features and incident-occurring environment features can significantly improve the effectiveness of DeepIP, confirming the contributions of these features.

6 DISCUSSION

Communication with Engineers in Microsoft. We have reported our results to the responsible engineers in Microsoft and communicated with them by emails. They expressed their troubles on incidental incidents, and also experienced and largely appreciated



Wilcoxon signed-rank test p-value:

AUC:
DeepIP v.s. DeepIP$_{noV}$: 0.0021
DeepIP v.s. DeepIP$_{noE}$: 0.0009

Precision:
DeepIP v.s. DeepIP$_{noV}$: 0.0025
DeepIP v.s. DeepIP$_{noE}$: 0.0004

Recall:
DeepIP v.s. DeepIP$_{noV}$: 0.0050
DeepIP v.s. DeepIP$_{noE}$: 0.0010

Figure 7: Contribution of each type of features

the functionality (i.e., prioritizing incidents by identifying incidental incidents) provided by us. For example, one engineer complained that she/he spent too much time on investigating false-alarm incidents, and noticed the real issue much later. Another engineer also pointed out that the current IcM system is overloaded and has many incidental incidents. In particular, they believed that the current performance of DeepIP is useful and gave us some suggestions to make our tool more user-friendly, e.g., it would be better if there is a "prioritized view" of incidents in the IcM system.

Generality of DeepIP. Although DeepIP is proposed and evaluated for incidents of online service systems, the framework of DeepIP is actually general. To investigate the generality of DeepIP, we applied it to predict the severity of traditional bug reports. In particular, we compared DeepIP with the state-of-the-art bug severity prediction approach [52] (introduced in Section 5.1.1). Here, we used the same *Mozilla* dataset released by the compared work, and used the results reported in their paper (the precision of 0.439 and the recall of 0.486). By applying DeepIP to the same dataset, DeepIP achieves the precision of 0.610 and the recall of 0.536, improving the state-of-the-art approach by 41.00% and 10.29% respectively. The result confirms the generality of DeepIP. The reason why DeepIP can perform well for bug reports is that, its attention mechanism can determine whether there are relevant bugs for the target bug, and it can understand semantics of textual descriptions, outperforming traditional text-similarity-based approaches. We also released the Mozilla dataset on our project webpage.

Impact of the Setting of Time Window. We investigated the impact of the time window on DeepIP. The default setting of the time window in DeepIP is 10. We also experimented different window sizes such as 0, 5, and 15. The results show that the default setting performs the best in general. The small settings (0 and 5) perform relatively worse than the large settings (10 and 15), indicating the necessity of considering a relatively large number of relevant incidents of a target incident. In addition, we conducted Wilcoxon signed-rank test at the significant level of 0.05 between the settings of 10 and 0 to investigate whether considering relevant incidents can significantly improve the effectiveness of DeepIP in terms of *AUC*, *Precision*, and *Recall*. The resulting p-values are all less than 0.05, confirming the contribution of relevant incidents in DeepIP.

Threats to Validity. The *internal* threat to validity mainly lies in the implementations of our approach and the compared approaches. To reduce this threat, two authors have carefully checked the code. For the various machine learning and information retrieval algorithms used in our work, we adopted the implementations provided by mature tools, which has been presented in Section 5.1.2.

The *external* threats to validity mainly lie in the subjects and compared approaches. In this work, we studied 18 real-world online service systems in Microsoft. All the used data are real in industry. To our best knowledge, this is the first large-scale study in this area. Even so, the used subjects may not represent systems in other companies. In the future, we will investigate more systems from different companies. Since the framework of DeepIP is general, it is easy to apply it to other companies as long as the companies have historical data for training. Also, some specific features may be different for different companies. In particular, we have conducted a study above by applying DeepIP to an open-source *Mozilla* dataset, demonstrating the generality of DeepIP. For the compared approaches, we selected two typical bug severity prediction approaches and also discussed the state-of-the-art approach [52] in Section 5.1.1. However, they may not represent other approaches. In the future, we will consider more approaches for comparison.

The *construct* threats to validity mainly lie in the used metrics, used parameters, and labeled data. To evaluate the performance of DeepIP, we used widely-used AUC, precision, and recall for measuring effectiveness, and used training time and predicting time for measuring efficiency. In the future, we will use more metrics to more sufficiently measure the performance of these approaches. For the parameters in the compared approaches, we set them using the values given in the papers or provided by the used mature tools. For the parameters in DeepIP, we set them via grid search, whose specific settings have been presented in Section 5.1.2. Also, we discussed the impact of the main parameter *time window size* above. In the future, we will further investigate the impact of other parameters. The incident data were labeled manually by engineers, and there may be noise. However, these engineers have rich experiences and domain knowledge, thus this threat may not be serious.

7 RELATED WORK

Incident Management. The most related work to ours is incident management. For example, Lou et al. [35, 36] presented an experience report on applying software analytics to incident management of online service systems, including incident diagnosis and mitigation. Chen et al. [10] conducted an extensive study to investigate incident triage for online service systems and evaluated the performance of traditional software bug triage techniques in the incident-triage context. Some work aims to associate a new incident with a previous known incident [17, 34]. For example, Duan and Babu [17] proposed an approach to improving the accuracy based on active learning, which maximizes the benefits gained from new unknown instances to facilitate manual labeling efforts. Different from them, our work aims to characterize incidents of online service systems and then prioritize incidents by identifying incidental incidents to improve the incident management process.

Bug Report Management. There are many common characteristics between incident reports of online service systems and bug reports of traditional software systems. Over the years, there have been many empirical studies on bug reports and bug-fixing performance [22, 30, 33, 38]. For example, Li et al. [33] manually collected

709 bugs from Mozilla and Apache Web Server, and analyzed the bug characteristics. Mockus et al. [38] proposed quality metrics (e.g., the percentage of defective files) to understand software maintenance efforts quantitatively. Guo et al. [22] performed an empirical study to characterize factors that determine which bugs get fixed in Windows 7. Some researchers also studied defect life cycles [15], bug distributions [4, 51], bug triage [5, 23], and bug-report based fault localization [48, 57]. Furthermore, there are also many papers on automatic assessment of the severity and priority of bug reports [32, 49, 52]. For example, Menzies and Marcus [37] proposed a tool named SEVERIS, which utilized standard information retrieval techniques and a rule learner to infer the connections between the most informative tokens in a bug report and the severity level. Tian et al. [44] proposed a machine learning based approach that can recommend a priority level based on information available in bug reports. Different from them, we perform an empirical study of *incidents* using industrial data and propose an approach to *prioritizing incidents*. Although incident and bug reports have much in common, they have some different characteristics. For example, bug reports are often reported and treated individually, while many incident reports tend to be correlated. One major reason is that an incident can lead to a series of other incidents, which can be detected by different monitors.

Empirical Analysis of Failures of Cloud Systems. Over the years, there have been many empirical studies on the characteristics of failures of data centers and cloud systems [20, 42, 45, 56]. For example, Schroeder and Gibson [42] described a large-scale study of failures in high-performance computing systems. Zhou et al.[56] performed an empirical study on quality issues of a real-world big data platform. Researchers also investigated the root causes of the system failures [21, 40, 50]. For example, Gray [21] found that administrator errors were responsible for 42% of system failures in high-end mainframes. Yin et al. [50] studied 546 real-world misconfigurations and found that a large portion of misconfigurations can cause hard-to-diagnose failures. Different from them, we focus on incidents rather than failures. Our study shows that not all incidents can lead to system failures. Many incidents (i.e., incidental incidents) are not important and will not get fixed with a high priority. In this work, we characterize the incidental incidents and propose a deep-learning based approach to prioritizing incidents. Our work allows engineers to more efficiently spend their efforts on incident management.

8 CONCLUSION

To better understand real-world incidents, we conduct a large-scale empirical study on incidents of 18 online service systems in Microsoft. We find that a large number of incidents are reported within a short period, but many incidents are incidental. Our qualitative and quantitative analysis show that on average, more than half of incidents are incidental and the percentage of maintenance time spent on them is up to 55.05%. Therefore, it is quite necessary to prioritize incidents by identifying incidental incidents in advance so as to optimize the incident management process. Towards this direction, we propose DeepIP, a deep learning based approach to prioritizing incidents by predicting the probabilities of incidents being incidental. Our experimental results show that DeepIP can achieve the AUC value of 0.808 on average with acceptable cost, which significantly outperforms all the compared approaches. In the future, we will further improve DeepIP to predict multiple categories of incidents instead of directly identifying incidental incidents and essential incidents.

ACKNOWLEDGEMENTS

This work was partially supported by ARC DP200102940.

REFERENCES

[1] Accessed: 2019. MXNet. https://mxnet.incubator.apache.org/

[2] Accessed: 2019. news. https://www.businessinsider.com/amazon-prime-day-website-issues-cost-it-millions-in-lost-sales-2018-7

[3] Accessed: 2019. scikit-learn. http://scikit-learn.org/stable/

[4] Carina Andersson and Per Runeson. 2007. A Replicated Quantitative Analysis of Fault Distributions in Complex Software Systems. *IEEE Trans. Softw. Eng.* 33, 5 (May 2007), 273–286. https://doi.org/10.1109/TSE.2007.1005

[5] John Anvik and Gail C Murphy. 2011. Reducing the effort of bug report triage: Recommenders for development-oriented decisions. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 20, 3 (2011), 10.

[6] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. *international conference on learning representations* (2015).

[7] Yoshua Bengio, Aaron Courville, and Pascal Vincent. 2013. Representation learning: A review and new perspectives. *IEEE transactions on pattern analysis and machine intelligence* 35, 8 (2013), 1798–1828.

[8] Michael W Berry and Malu Castellanos. 2004. Survey of text mining. *Computing Reviews* 45, 9 (2004), 548.

[9] David M Blei, Andrew Y Ng, and Michael I Jordan. 2003. Latent dirichlet allocation. *Journal of machine Learning research* 3, Jan (2003), 993–1022.

[10] 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 ACM/IEEE International Conference on Software Engineering*. to appear.

[11] 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.

[12] 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 *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. to appear.

[13] Stéphan Clémençon and Nicolas Vayatis. 2007. Ranking the best instances. *Journal of Machine Learning Research* 8, Dec (2007), 2671–2699.

[14] Corinna Cortes and Mehryar Mohri. 2003. AUC Optimization vs. Error Rate Minimization. In *Proceedings of the 16th International Conference on Neural Information Processing Systems*. 313–320.

[15] Marco D'Ambros, Michele Lanza, and Martin Pinzger. 2007. " A Bug's Life" Visualizing a Bug Database. In *Visualizing Software for Understanding and Analysis, 2007. VISSOFT 2007. 4th IEEE International Workshop on*. IEEE, 113–120.

[16] Min Du, Feifei Li, Guineng Zheng, and Vivek Srikumar. 2017. Deeplog: Anomaly detection and diagnosis from system logs through deep learning. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 1285–1298.

[17] Songyun Duan and Shivnath Babu. 2008. Guided problem diagnosis through active learning. In *International Conference on Autonomic Computing*. 45–54.

[18] Tom Fawcett. 2006. An introduction to ROC analysis. *Pattern Recognition Letters* 27, 8 (2006), 861–874.

[19] George Forman and Martin Scholz. 2010. Apples-to-apples in cross-validation studies: pitfalls in classifier performance measurement. *ACM SIGKDD Explorations Newsletter* 12, 1 (2010), 49–57.

[20] Phillipa Gill, Navendu Jain, and Nachiappan Nagappan. 2011. Understanding network failures in data centers: measurement, analysis, and implications. In *ACM SIGCOMM Computer Communication Review*, Vol. 41. ACM, 350–361.

[21] Jim Gray. 1986. Why do computers stop and what can be done about it?. In *Symposium on reliability in distributed software and database systems*. Los Angeles, CA, USA, 3–12.

[22] Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and predicting which bugs get fixed: an empirical study of Microsoft Windows. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering-Volume 1*. ACM, 495–504.

[23] Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2011. Not my bug! and other reasons for software bug report reassignments. In

Proceedings of the ACM 2011 conference on Computer supported cooperative work. ACM, 395–404.

[24] David J Hand. 2007. Principles of data mining. *Drug safety* 30, 7 (2007), 621–622.

[25] David Harris and Sarah Harris. 2012. *Digital Design and Computer Architecture, Second Edition* (2nd ed.). Morgan Kaufmann Publishers Inc.

[26] 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.

[27] Rie Johnson and Tong Zhang. 2017. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Vol. 1. 562–570.

[28] Armand Joulin, Edouard Grave, Piotr Bojanowski, Matthijs Douze, Hérve Jégou, and Tomas Mikolov. 2016. FastText.zip: Compressing text classification models. *arXiv preprint arXiv:1612.03651* (2016).

[29] Yujun Chen; Xian Yang; Qingwei Lin; Hongyu Zhang; Feng Gao; Zhangwei Xu; Yingnong Dang; Dongmei Zhang; Hang Dong; Yong Xu; Hao Li; Yu Kang;. 2019. Outage Prediction and Diagnosis for Cloud Service Systems. In *Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018.* to appear.

[30] Sunghun Kim and E James Whitehead Jr. 2006. How long did it take to fix bugs?. In *Proceedings of the 2006 international workshop on Mining software repositories*. ACM, 173–174.

[31] Yoon Kim. 2014. Convolutional Neural Networks for Sentence Classification. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, A meeting of SIGDAT, a Special Interest Group of the ACL*. 1746–1751.

[32] Ahmed Lamkanfi, Serge Demeyer, Emanuel Giger, and Bart Goethals. 2010. Predicting the severity of a reported bug. In *Mining Software Repositories (MSR), 2010 7th IEEE Working Conference on*. IEEE, 1–10.

[33] Zhenmin Li, Lin Tan, Xuanhui Wang, Shan Lu, Yuanyuan Zhou, and Chengxiang Zhai. 2006. Have things changed now?: an empirical study of bug characteristics in modern open source software. In *Proceedings of the 1st workshop on Architectural and system support for improving software dependability*. ACM, 25–33.

[34] Meng Hui Lim, Jian Guang Lou, Hongyu Zhang, Fu Qiang, Andrew Teoh, Qingwei Lin, Justin Ding, and Dongmei Zhang. 2015. Identifying Recurrent and Unknown Performance Issues. In *IEEE International Conference on Data Mining*.

[35] Jianguang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2013. Software analytics for incident management of online services: an experience report. *automated software engineering* (2013), 475–485.

[36] Jianguang Lou, Qingwei Lin, Rui Ding, Qiang Fu, Dongmei Zhang, and Tao Xie. 2017. Experience report on applying software analytics in incident management of online service. *automated software engineering* 24, 4 (2017), 905–941.

[37] Tim Menzies and Andrian Marcus. 2008. Automated severity assessment of software defect reports. In *Software Maintenance, 2008. ICSM 2008. IEEE International Conference on*. IEEE, 346–355.

[38] Audris Mockus, Roy T Fielding, and James D Herbsleb. 2002. Two case studies of open source software development: Apache and Mozilla. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 11, 3 (2002), 309–346.

[39] Nasser M Nasrabadi. 2007. Pattern recognition and machine learning. *Journal of electronic imaging* 16, 4 (2007), 049901.

[40] David Patterson, Aaron Brown, Pete Broadwell, George Candea, Mike Chen, James Cutler, Patricia Enriquez, Armando Fox, Emre Kiciman, Matthew Merzbacher, et al. 2002. *Recovery-oriented computing (ROC): Motivation, definition, techniques, and case studies.* Technical Report. Technical Report UCB//CSD-02-1175, UC Berkeley Computer Science.

[41] Stephen Robertson, Hugo Zaragoza, et al. 2009. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends® in Information Retrieval* 3, 4 (2009), 333–389.

[42] Bianca Schroeder and Garth Gibson. 2010. A large-scale study of failures in high-performance computing systems. *IEEE Transactions on Dependable and Secure Computing* 7, 4 (2010), 337–350.

[43] Yuan Tian, David Lo, and Chengnian Sun. 2013. Drone: Predicting priority of reported bugs by multi-factor analysis. In *2013 IEEE International Conference on Software Maintenance*. IEEE, 200–209.

[44] Yuan Tian, David Lo, Xin Xia, and Chengnian Sun. 2015. Automated prediction of bug report priority using multi-factor analysis. *Empirical Software Engineering* 20, 5 (2015), 1354–1383.

[45] Kashi Venkatesh Vishwanath and Nachiappan Nagappan. 2010. Characterizing cloud computing hardware reliability. In *Proceedings of the 1st ACM symposium on Cloud computing*. ACM, 193–204.

[46] Lingzhi Wang, Nengwen Zhao, Junjie Chen, Pinnong Li, Wenchi Zhang, and Kaixin Sui. 2020. Root-Cause Metric Location for Microservice Systems via Log Anomaly Detection. In *The 2020 IEEE International Conference on Web Services*. to appear.

[47] Frank Wilcoxon. 1945. Individual Comparisons by Ranking Methods. *Biometrics Bulletin* 1, 6 (1945), 80–83.

[48] Chu-Pan Wong, Yingfei Xiong, Hongyu Zhang, Dan Hao, Lu Zhang, and Hong Mei. 2014. Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In *2014 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 181–190.

[49] Cheng-Zen Yang, Chun-Chi Hou, Wei-Chen Kao, and Xiang Chen. 2012. An empirical study on improving severity prediction of defect reports using feature selection. In *Software Engineering Conference (APSEC), 2012 19th Asia-Pacific*, Vol. 1. IEEE, 240–249.

[50] Zuoning Yin, Xiao Ma, Jing Zheng, Yuanyuan Zhou, Lakshmi N Bairavasundaram, and Shankar Pasupathy. 2011. An empirical study on configuration errors in commercial and open source systems. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*. ACM, 159–172.

[51] H. Zhang. 2008. On the Distribution of Software Faults. *IEEE Transactions on Software Engineering* 34, 2 (March 2008), 301–302. https://doi.org/10.1109/TSE.2007.70771

[52] Tao Zhang, Jiachi Chen, Geunseok Yang, Byungjeong Lee, and Xiapu Luo. 2016. Towards more accurate severity prediction and fixer recommendation of software bugs. *Journal of Systems and Software* 117 (2016), 166–184.

[53] Xu Zhang, Yong Xu, Qingwei Lin, Bo Qiao, Hongyu Zhang, Yingnong Dang, Chunyu Xie, Xinsheng Yang, Qian Cheng, Ze Li, et al. 2019. Robust log-based anomaly detection on unstable log data. In *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. 807–817.

[54] Nengwen Zhao, Junjie Chen, Xiao Peng, Honglin Wang, Xinya Wu, Yuanzong Zhang, Zikai Chen, Xiangzhong Zheng, Xiaohui Nie, Gang Wang, Yong Wu, Fang Zhou, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Understanding and Handling Alert Storm for Online Service Systems. In *The 42nd International Conference on Software Engineering, SEIP track.* to appear.

[55] Nengwen Zhao, Junjie Chen, Zhou Wang, Xiao Peng, Gang Wang, Yong Wu, Fang Zhou, Zhen Feng, Xiaohui Nie, Wenchi Zhang, Kaixin Sui, and Dan Pei. 2020. Real-time Incident Prediction for Online Service Systems. In *The 28th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*. to appear.

[56] Hucheng Zhou, Jian-Guang Lou, Hongyu Zhang, Haibo Lin, Haoxiang Lin, and Tingting Qin. 2015. An Empirical Study on Quality Issues of Production Big Data Platform. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2* (Florence, Italy) *(ICSE '15)*. IEEE Press, Piscataway, NJ, USA, 17–26. http://dl.acm.org/citation.cfm?id=2819009.2819014

[57] Jian Zhou, Hongyu Zhang, and David Lo. 2012. Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports. In *Software Engineering (ICSE), 2012 34th International Conference on*. IEEE, 14–24.