Continuous Incident Triage for Large-Scale Online Service Systems

Junjie Chen[1], Xiaoting He[2], Qingwei Lin[2], Hongyu Zhang[3], Dan Hao[4,5],
Feng Gao[6], Zhangwei Xu[6], Yingnong Dang[6], Dongmei Zhang[2]
[1]College of Intelligence and Computing, Tianjin University, Tianjin, China, junjiechen@tju.edu.cn
[2]Microsoft Research, Beijing, China, {v-xiah,qlin,dongmeiz}@microsoft.com
[3]The University of Newcastle, NSW, Australia, hongyu.zhang@newcastle.edu.au
[4]Key Laboratory of High Confidence Software Technologies (Peking University), MoE
[5]Department of Computer Science and Technology, EECS, Peking University, Beijing, China, haodan@pku.edu.cn
[6]Microsoft Azure, Redmond, Washington, USA {fgao,zhangxu,yidang}@microsoft.com

***Abstract*—In recent years, online service systems have become increasingly popular. Incidents of these systems could cause significant economic loss and customer dissatisfaction. Incident triage, which is the process of assigning a new incident to the responsible team, is vitally important for quick recovery of the affected service. Our industry experience shows that in practice, incident triage is not conducted only once in the beginning, but is a *continuous process*, in which engineers from different teams have to discuss intensively among themselves about an incident, and continuously refine the incident-triage result until the correct assignment is reached. In particular, our empirical study on 8 real online service systems shows that the percentage of incidents that were reassigned ranges from 5.43% to 68.26% and the number of discussion items before achieving the correct assignment is up to 11.32 on average. To improve the existing incident triage process, in this paper, we propose DeepCT, a Deep learning based approach to automated Continuous incident Triage. DeepCT incorporates a novel GRU-based (Gated Recurrent Unit) model with an attention-based mask strategy and a revised loss function, which can incrementally learn knowledge from discussions and update incident-triage results. Using DeepCT, the correct incident assignment can be achieved with fewer discussions. We conducted an extensive evaluation of DeepCT on 14 large-scale online service systems in Microsoft. The results show that DeepCT is able to achieve more accurate and efficient incident triage, e.g., the average accuracy identifying the responsible team precisely is 0.641∼0.729 with the number of discussion items increasing from 1 to 5. Also, DeepCT statistically significantly outperforms the state-of-the-art bug triage approach.**

***Index Terms*—Incident Triage, Online Service Systems, Deep Learning**

I. Introduction

Online service systems have become increasingly popular in recent years [1]–[3]. For example, Microsoft Office 365 has about 120 millions of monthly active users in October 2017[1]. Although tremendous efforts have been devoted to assure the quality of online service systems, these systems still encounter many incidents (i.e., unplanned interruptions and outages of the service), which cause huge economic loss or dramatically decrease user satisfaction. For example, a study conducted on 63 data center organizations in the U.S. shows that the average cost of service downtime has steadily increased from $505,502 in 2010 to $740,357 in 2016[2]. Also, the one-hour downtime for `Amazon.com` on Prime Day last year (its biggest sale event of the year) leads to the loss of up to $100 million[3].

Once an incident occurs to an online service system, it should be mitigated as soon as possible, in order to minimize the service downtime and ensure high availability of the provided service. One important step of incident handling is incident triage, which is the assignment of a new incident to the responsible team. Incident triage is critical: if an incident is assigned to a wrong team, the mitigation of the incident could be delayed and more loss could be incurred.

However, accurate and efficient incident triage is very challenging for large-scale online service systems. One major reason is that unlike traditional software bugs [4]–[6], a large portion of incidents are automatically reported by monitors rather than people. For example, the existing study demonstrated that more than 90% incidents are automatically reported by monitors for online service systems in Microsoft [1]. These automatically reported incidents are created based on certain simple templates and do not contain detailed textual descriptions about the incidents. That is, an incident report usually provides limited textual information for engineers to understand the incident, and thus it is difficult for them to accurately and efficiently assign the incident to the responsible team.

Our industry experience shows that in practice, engineers have to discuss intensively among themselves about an incident, and continuously refine the incident-triage result during discussions until the correct assignment is reached. During this

This work is supported by the National Natural Science Foundation of China under Grant No. 61872008, 61828201, 61872263, U1836214. This work was mainly done when Junjie Chen was visiting Microsoft Research. Qingwei Lin is the corresponding author.

[1] https://www.zdnet.com/article/microsoft-office-365-now-has-120-million-business-users/.

[2] https://www.ponemon.org/blog/2016-cost-of-data-center-outages.

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

process, more engineers from different teams could join the discussions and more information could be provided, leading to gradually approach the correct triage. That is, incident triage in practice is not conducted only once in the beginning, but is a continuous process. To better understand the incident triage practice, we conducted an empirical study on 8 real online service systems in Microsoft (Section II). The results show that the percentage of incidents that were reassigned (assigned at least twice) ranges from 5.43% to 68.26% for the 8 online service systems. For these incidents, the number of discussion items before achieving the correct assignment is up to 11.32 on average. That is, in practice, many incidents are indeed incorrectly assigned in the beginning and engineers have to spend significant efforts on discussions to refine incident triage. In this paper, we call this scenario *continuous incident triage*.

To address the problem of continuous incident triage, in this paper we propose the first automated approach, called **DeepCT** (**Deep** learning based **C**ontinuous incident **T**riage). DeepCT incorporates a novel GRU-based (Gated Recurrent Unit [7]) model with an attention-based mask strategy and a revised loss function, which can incrementally learn knowledge from discussions and update incident-triage results. Since discussions are conducted by engineers manually like conversations, it tends to introduce noise during actual discussions. The attention-based mask strategy can help reduce the impact of noise by automatically assigning them quite small weights. Furthermore, we propose a domain-specific text encoding method to conduct semantic understanding for textual information by considering the text characteristics of online service systems. Using DeepCT, the correct incident assignment can be achieved with as few rounds of discussions as possible.

To evaluate the effectiveness of DeepCT, we conducted an extensive study on 14 industrial large-scale online service systems in Microsoft. These studied systems have millions or even hundreds of millions of users around the world. The experimental results demonstrate that DeepCT is able to achieve more accurate and efficient incident triage in the real-world scenario of continuous incident triage. More specifically, DeepCT achieves the average accuracy (identifying the responsible team precisely) of 0.641∼0.729 when the number of discussion items increases from 1 to 5. Also, DeepCT statistically significantly outperforms the state-of-the-art bug-triage approach [8], which has been demonstrated to be the most effective approach in the incident-triage context before [1]. Furthermore, the results also show that each component (i.e., the attention-based mask strategy and the revised loss function) indeed contributes to DeepCT.

To sum up, our work has the following major contributions:

- We are the first to identify continuous incident triage, a common scenario of incident triage for large-scale online service systems in practice. We have conducted an empirical study to investigate this scenario.
- We propose DeepCT, the first automated approach to solving the problem of continuous incident triage based on deep learning.
- We conduct an extensive study based on 14 industrial large-scale online service systems in Microsoft. The results demonstrate the effectiveness of DeepCT.

This paper is organized as follows: Section II presents an empirical study that motivates our work; Section III describes the details of DeepCT; Section IV reports the experiments and corresponding results; Section V discusses the lessons learned from our industry experience, the generality of DeepCT, and threats to validity; Section VI presents the related work; Section VII concludes the paper.

II. An Empirical Study on Incident Triage

Since it is the first time to systematically investigate continuous incident triage for large-scale online service systems, we conducted an empirical study on this scenario based on 8 industrial large-scale online service systems in Microsoft. In particular, we collected six months of incident data for the 8 studied systems. Please note that we collected only the incidents that have been resolved in the study. Due to the company policy, we have to hide some details of these incidents and report relatively rough data instead.

In this study, we focus on the following two aspects: ① *the accuracy of incident triage in the beginning*; ② *the engineers' efforts on refining incident triage*.

A. Accuracy of Incident Triage in the Beginning

Incident triage in the beginning tends to be manually conducted by On-Call Engineers (OCEs) who have rich experience and domain knowledge. They manually assign incidents to the responsible team based on the information provided by incident reports. Therefore, it is interesting to investigate how such manual incident triage performs in practice, i.e., the accuracy of incident triage in the beginning.

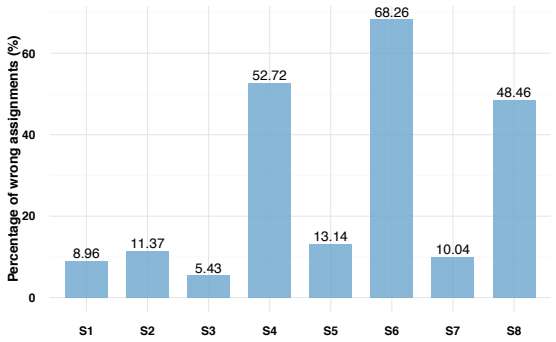

Fig. 1: Percentage of wrong assignments in the beginning

Figure 1 shows the percentage of wrong assignments in the beginning through manual incident triage. In this figure, the number above each bar represents the percentage for the corresponding system. From this figure, the percentage of wrong assignments in the beginning for the 8 studied systems ranges from 5.43% to 68.26%. The average percentage of them is 27.30%, which means that, even though the OCEs have rich experience and domain knowledge, they still make many

mistakes during incident triage in the beginning. One major reason is that the incident reports often provide them very limited information.

B. Engineers' Efforts on Refining Incident Triage

When OCEs make a mistake for incident triage in the beginning, engineers from various product teams conduct intensive discussions about the incident to revise the assignment of the incident. In this study, we also analyzed the engineers' efforts on refining incident triage. Here we used two complementary metrics to measure engineers' efforts, namely the number of discussion items and the time spent on discussions for achieving correct triage.

We first investigated the number of discussion items before achieving correct triage for the incidents that are incorrectly assigned in the beginning. The goal is to understand how intensively the discussions are conducted among engineers in order to achieve correct triage. Here we regard the discussion items occurred before the final triage (i.e., correct triage) is made as the *triage-stage* discussion items.

TABLE I: Number and percentage of triage-stage discussion items

Sys.	S1	S2	S3	S4	S5	S6	S7	S8
Num.	8.45	20.55	10.75	15.81	6.42	8.59	13.56	6.42
Per.(%)	52.29	55.53	78.41	38.26	51.40	74.18	47.31	62.27

The second row in Table I presents the average number of triage-stage discussion items for each online service system. We find that engineers usually need to conduct 6.42∼20.55 discussion items before achieving correct triage. The average number of triage-stage discussion items for the 8 systems is 11.32. That is, the discussions among engineers for refining incident triage are indeed intensive, indicating that engineers spend non-trivial efforts on refining incident triage for large-scale online service systems.

Besides discussions for refining incident triage, engineers also conduct follow-up discussions to determine how to mitigate incidents. Therefore, it is also interesting to learn what percentage of triage-stage discussion items in all discussion items about the incidents. The last row in Table I presents the percentage for each online service system. We find that around 38.26%∼78.41% discussion items are conducted for incident triage. The average percentage of them is 57.46%. That is, more than half of discussion items are conducted to refine incident triage, indicating that the efforts on refining incident triage are no less than those on follow-up incident mitigation.

We then investigated the percentage of the time spent on triage-stage discussions among the time spent on overall discussions about the incidents (i.e., the time from incident creation to incident mitigation, call TTM). Here, each discussion item is created along with a timestamp and each activity (such as incident triage, mitigation, and resolution) is also recorded along with a timestamp in the incident management system. We calculated the above two types of time based on these timestamps. This calculation method may be a threat, since it is hard to ensure developers to manage an incident all the time until the incident is mitigated. Nevertheless, the percentage is still able to complement the analysis on the number of discussion items to some degree.

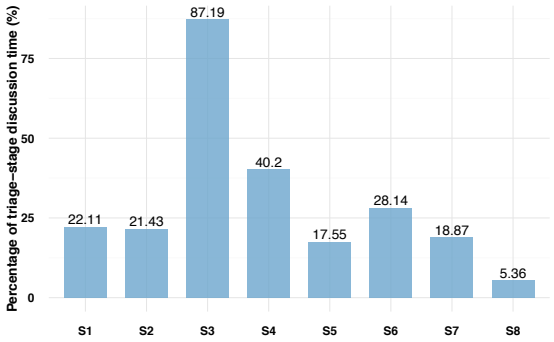

Fig. 2: Percentage of discussion time for incident triage within the overall discussion time (TTM)

Figure 2 shows the percentage of discussion time for incident triage within the overall TTM. In this figure, the number above each bar represents the percentage for each studied system. From this figure, the percentage of discussion time on incident triage ranges from 5.36% to 87.19% within the overall TTM. Their average percentage is 30.11%. That is, the discussion time spent on incident triage is indeed non-trivial compared with that spent on the whole incident mitigation process (from incident creation to incident mitigation). This result further supports the conclusion made based on the number of discussion items for incident triage.

C. Summary

Based on the empirical study, we obtain the following two main findings:

- Many incidents are actually incorrectly assigned in the beginning due to the limited information provided by the incident reports.
- Engineers indeed have to spend significant amount of efforts on discussions about incident triage.

These findings support the scenario of incident triage for large-scale online service systems, i.e., continuous incident triage. Therefore, it is necessary to propose an effective incident-triage approach, which can continuously refine incident assignments based on incrementally provided discussions.

III. Approach

A. Overview

In this work, we target at the problem of continuous incident triage. As described in previous sections, in practice, discussions are incrementally created by engineers. However, existing bug-triage approaches for traditional software systems either ignore discussions or simply treat all discussions as a whole without considering their characteristics, i.e., *incremental* creation. Therefore, these existing approaches cannot work well in this real-world scenario. To effectively conduct

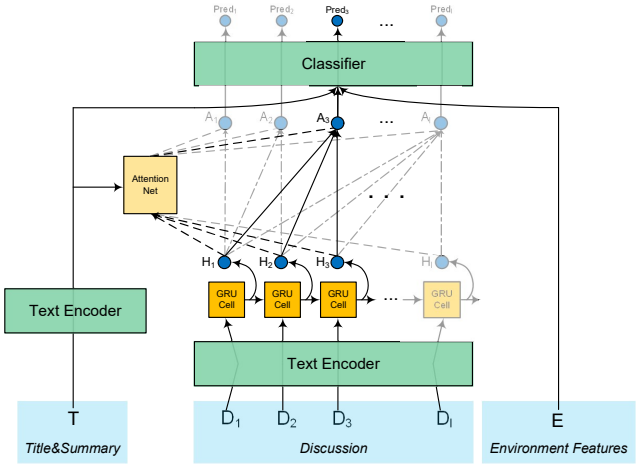

Fig. 3: Overview of DeepCT

continuous incident triage, one major challenge is how to learn knowledge from incremental discussions to fit the scenario of continuous incident triage. Moreover, since discussions are conducted by engineers manually like conversations, it tends to introduce noise during actual discussions. Therefore how to reduce the impact of noise on incident triage is another challenge.

To solve the problem of continuous incident triage, we propose the first automated approach, **DeepCT**, a deep learning based approach that can incrementally learn knowledge from incident reports. The goal is to achieve correct incident-triage results with as few discussions as possible. More specifically, to solve the first challenge, we design a GRU-based model with a revised loss function in DeepCT to effectively utilize incremental discussions by considering their temporal relations so that correct incident-triage results can be achieved as early as possible. To solve the second challenge, we propose an attention-based mask strategy in the model to bypass noise. In this way, different weights can be automatically assigned to different discussion information, where noise can be masked by assigning them quite small weights. Furthermore, since DeepCT is mainly based on textual information such as textual discussions, we propose a domain-specific text encoding method to process textual information by considering the text characteristics of online service systems. The overview of DeepCT is shown in Figure 3.

In the following, we first introduce the input data used in DeepCT in Section III-B. Then, we introduce our domain-specific text encoding method in Section III-C. Next, we introduce our designed GRU-based model, including our attention-based mask strategy, in Section III-D. Finally, we introduce the usage of DeepCT in Section III-E.

B. Input Data

We consider three types of input data in DeepCT for continuous incident triage. The first type of input data used in DeepCT is *the title and summary of an incident report*, which is widely used in traditional bug triage. The title and summary of an incident report are the textual description about the symptom when an incident is reported. Engineers usually conduct incident triage manually in the beginning based on this type of information.

The second type of input data used in DeepCT is *the incremental discussions about an incident*. The discussions are textual information, which are manually written by engineers incrementally like conversations. As presented in Section II, engineers tend to spend significant efforts on discussions for refining incident triage in practice. That is, this type of input data is the core information for continuous incident triage. In particular, different from existing bug-triage approaches (i.e., ignoring discussions or simply treating all discussions as a whole), DeepCT utilizes the type of input data *incrementally* to effectively fit the scenario of continuous incident triage.

The third type of input data used in DeepCT is *the incident-occurring environment information*. An incident tends to be reported with the incident-occurring environment information, including the monitor ID reporting the incident, the incident-occurring device, and the incident type (how to report the incident such as monitor reporting or human reporting). Engineers also utilize this type of information to facilitate incident management.

C. Domain-specific Text Encoding

Since the first and second types of information are textual descriptions, DeepCT encodes them into vectors to conduct semantic understanding. However, there are many special terms about online service systems in textual descriptions, such as API names and component names, which are helpful to conduct incident triage but cannot be well handled by traditional text encoding methods due to the small occurrence frequency of each special term. More specifically, traditional text encoding methods either ignore these special terms or treat them as "unknown word" uniformly [9]. Therefore, we propose a domain-specific text encoding method in DeepCT by considering the text characteristics of online service systems.

The domain-specific text encoding method first uses the FastText algorithm [9] to build pre-trained subword vectors based on external corpus, and then conducts fine tuning based on historical incident data to incorporate the domain knowledge of online service systems. Finally, based on the pre-trained subword vectors, the representation of each special term can be learned by integrating the representations of the subwords of the special term. In this way, the title&summary or each discussion item is transformed into a matrix, in which the number of rows is equal to the number of words in the corresponding textual description.

Next, the method encodes a matrix into a vector using a CNN-based neural-language model, which has been demonstrated to be able to encode more complex patterns and focus on word-level knowledge to produce better performance [10], [11]. More specifically, it first uses an average pooling layer and a convolution layer to merge adjacent words (i.e., rows in the matrix) and then produces feature maps from the

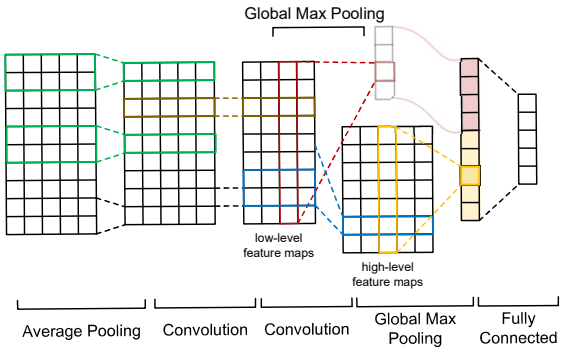

Fig. 4: CNN-based Text Encoder

matrix, which is called *low-level* feature maps in this paper. To capture the correlations among words with farther distances, it further uses a convolution layer to produce *high-level* feature maps based on these low-level ones. Since both low-level feature maps and high-level feature maps can provide useful information, it then uses two global max pooling layers to produce feature vectors for low-level feature maps and high-level feature maps, respectively, in order to extract the most important words. Finally, based on these produced vectors, it uses a fully connected layer to generate a vector for the textual description. In this way, the title&summary or each discussion item of an incident is encoded into a vector. We denote the vector of the title&summary as $T = \{t_1, t_2, \ldots, t_r\}$, and the vector of the i_{th} discussion item as $D_i = \{d_{i1}, d_{i2}, \ldots, d_{ir}\}$, where r is the dimension of the fully connected layer. Figure 4 shows the CNN-based text encoder in DeepCT.

In addition, since the values of each input datum in the third type is a finite set of discrete values, we conduct feature embedding for it. Here, we adopt the representation learning [12] to embed each input-datum value into a vector in DeepCT, which is able to embed a value to a fixed-dimension vector and gradually updates the vector during the training process. In this way, each input datum in the third type is embedded into a fixed-dimension vector. We denote the k_{th} feature vector as $E_k = \{e_{k1}, e_{k2}, \ldots, e_{ks}\}$, where s refers to the pre-defined fixed dimension in representation learning for the third type of input data. Here, we concatenate all the feature vectors for the third type into a vector $E = E_1 \oplus E_2 \oplus \ldots \oplus E_z$, where z is the total number of feature vectors for the third type of input data, and $\oplus$ is the concatenation operator.

D. Designed GRU-based Model

To conduct continuous incident triage, we design a GRU-based model, which is able to enhance the learning for the knowledge from earlier discussions so that correct assignments can be achieved with fewer discussions. Since there exist temporal relations among discussion items, our designed model is built on the GRU (Gated Recurrent Unit) network [7], which is presented in Section III-D1. We also propose an attention-based mask strategy in our designed model to reduce the impact of noise, which is presented in Section III-D2. To help achieve the goal of correct incident-triage results with as few discussions as possible, we further propose a continuous loss function in our model, which is presented in Section III-D3.

1) GRU Network: GRU is one of the most widely-used network to solve the problem involving temporal relations [7]. In our context, discussions are incrementally created by engineers in chronological order, and thus we adopt the GRU network to build our model. More specifically, GRU has two gates: update gate and reset gate. The former is to determine how much past information (from previous discussion items) to pass along to the future, while the latter is to determine how much past information to forget.

For an incident, the input of GRU at the t_{th} time step is the vector of the t_{th} discussion item denoted as D_t, where $1 \leq t \leq n$ and n refers to the total number of discussion items. Given D_t and the hidden state from the $(t-1)_{th}$ time step, denoted as H_{t-1}, GRU calculates the update gate and reset gate by Formula 1, where r_t and z_t are the update gate and reset gate at the t_{th} time step, respectively. Here, the hidden state H_{t-1} contains the retained information from previous $t-1$ discussion items at the $(t-1)_{th}$ time step. Please note that H_0 is a zero vector since there is no discussion at the 0_{th} time step.

$$\begin{aligned} r_t &= \sigma(W_r \cdot [H_{t-1}, D_t]) \\ z_t &= \sigma(W_z \cdot [H_{t-1}, D_t]) \end{aligned} \tag{1}$$

After acquiring them, GRU uses the reset gate to forget some past information and then combine the retained past information with new information, which is calculated by Formula 2. Then, GRU uses the update gate to decide what to collect from the combined information and what to collect from past information at the current time step, which is calculated by Formula 3. That is, the output of GRU at the t_{th} time step is the hidden state H_t, which contains the retained information from previous t discussion items at the t_{th} time step. Please note that W_r, W_z, and $W_{\tilde{h}}$ are parameters that are learned during the training process.

$$\tilde{H}_t = tanh(W_{\tilde{h}} \cdot [r_t \odot H_{t-1}, D_t]) \tag{2}$$

$$H_t = z_t \cdot H_{t-1} + (1 - z_t) \cdot \tilde{H}_t \tag{3}$$

2) Attention-based Mask Strategy: At the t_{th} time step (receiving the t_{th} discussion item), t hidden states are produced from the GRU network, i.e., $H_1, \ldots, H_t$. Since noise may be introduced during discussions, we incorporate an attention mechanism to automatically learn the weights of $H_1, \ldots, H_t$. In particular, the title&summary of an incident report describes the incident with relative high relevance. Therefore, if the correlation between the title&summary and the discussion information H_i $(1 \leq i \leq t)$ is larger, the weight of H_i should be larger, indicating that H_i contains less noise. In this way, noise can be masked by assigning them quite small weights.

Here, we denoted the attention-based vector integrated by $H_1, \ldots, H_t$ as A_t, which is calculated by Formula 4.

$$A_t = \sum_{i=1}^{t} \omega_{ti} * H_i \tag{4}$$

where, ω_{ti} is the learned weight of H_i at the t_{th} time step. More specifically, ω_{ti} is calculated by the *softmax* function, which is shown in Formula 5. In this formula, $f(T, H_i) = v^\top MLP(T \oplus H_i)$, $\oplus$ is the concatenation operator, and v is a parameter learned in MLP [13]. Please note that the weights are recalculated at a new time step. That is, the same hidden state has different weights at different time steps.

$$\omega_{ti} = \frac{e^{f(T,H_i))}}{\sum_{k=1}^{t} e^{f(T,H_k))}} \tag{5}$$

3) Continuous Loss Function: After acquiring all the attention-based vectors $A_1 \ldots A_l$ where l is the total number of discussion items, DeepCT utilizes them and the other two feature vectors (i.e., T and E) to predict the incident-triage result. More specifically, DeepCT uses MLP (with a softmax layer) for classification, where the input of MLP is $A_1 \ldots A_l$, T and E. The output is the predicted probability that each team is the responsible one. Since our goal is to achieve the correct incident-triage result as much as possible at each time step, we propose a continuous loss function. Instead of calculating the loss (i.e., cross entropy) at the last time step, this function calculates the sum of the loss at each time step (from the 1_{st} to l_{th} time steps). In this way, the cumulative prediction results at each time step can be optimized when calculating gradient during back propagation. More specifically, the continuous loss function is shown in Formula 6.

$$Loss(t) = \sum_{k=1}^{n} \left[\sum_{j=1}^{l} \left(- \sum_{i=1}^{m} c_i \log \hat{c}_i^{(k)} \right) \right] \tag{6}$$

where, c_i is the predicted probability that the i_{th} team is the responsible one at the j_{th} time step, m is the total number of teams, and n is the total number of incidents.

E. Usage of DeepCT

We first use DeepCT to build a classification model based on historical incident data, and then use the model to predict the responsible team for each new incident. More specifically, for each new incident, DeepCT first constructs T and E, and takes them as input of the model to predict the responsible team, which is the incident triage in the beginning. Then, when a new triage-stage discussion item (the i_{th} discussion item) is created, DeepCT constructs D_i and incrementally passes it to the model to predict the responsible team. That is, DeepCT conducts incident triage continuously when the discussions are going on. In particular, the model at each triage can predict the probability to be responsible for each team, and its output is the team with the largest probability.

IV. Evaluation

In the study, we address the following research questions:

- **RQ1**: How does DeepCT perform in continuous incident triage for large-scale online service systems?
- **RQ2**: Does each main component contribute to DeepCT?
- **RQ3**: What is the time efficiency of DeepCT?

A. Study Data

In the study, we applied DeepCT to 14 industrial online service systems in Microsoft, to sufficiently evaluate the effectiveness of DeepCT. These systems are large and complex online service systems, which have been widely used by millions of users around the world. All the 14 systems are in different application areas and developed by different product groups, indicating the diversity of these systems. We collected six months of incident data for each studied system, and the total size of the incident reports is over 90GB. The total number of teams is about 2K. Therefore, it is challenging to assign an incident to the correct team. Please note that in our experiments, we only consider the incidents that have been resolved, since the correct assignments of these incidents have been achieved and thus they can help evaluate the effectiveness of incident triage. In particular, we used the incident data from the former four months as training data, and used the incident data from the latter two months as testing data to evaluate DeepCT.

B. Compared Approaches

1) Baselines: According to the existing study [1], which investigated how existing bug-triage approaches for traditional software perform in the incident-triage context for online service systems, the deep-learning-based bug-triage approach [8] performs the best in the context (outperforming the machine-learning-based and information-retrieval-based bug-triage approaches). Therefore, in this study we used this approach as the representative for comparison. More specifically, this approach first converts each word in the textual description of an incident report to a vector representation using Word2Vec [14], and then adopts Convolutional Neural Network (CNN) [15] to train a classifier for recommending responsible teams of new incidents. Based on the way of dealing with discussions during training, this approach can have two forms. The first one ignores discussions, and only considers title&summary as the textual description in each incident report during training. We call it **DL**$_{no}$ in this paper. The second one treats all discussions as a whole, and considers them and title&summary as the textual description during training. We call it **DL**$_{all}$ in this paper. In summary, there are two baselines in the study, i.e., DL$_{no}$ and DL$_{all}$. In particular, we also adapted them to refine incident triage (i.e., conduct a new triage) when a new discussion item is created.

2) Variants of DeepCT: In RQ2, we aim to investigate the contributions of two main components in DeepCT, i.e., the attention-based mask strategy and the continuous loss function. To investigate the contribution of the first component, we constructed a variant of DeepCT by removing the attention-based

mask strategy from DeepCT, which is called **DeepCT**$^{mask}_{no}$. That is, after acquiring $H_1, \ldots, H_t$, DeepCT$^{mask}_{no}$ directly takes them and T and E as input of MLP for classification. To investigate the contribution of the second component, we constructed another variant of DeepCT by replacing the continuous loss function with directly calculating the loss at the last time step. This variant is called **DeepCT**$^{loss}_{ori}$.

C. Implementation

Our approach DeepCT and the compared approaches are implemented based on Apache MXNet[4], a scalable deep learning framework. For the hyperparameters in these approaches, we tuned them through grid search following the existing work [10]. More specifically, for the hyperparameters in CNN, we set them as follows: the word-embedding size is 300, the pool size and stride of the average pooling layer are 2 and 1 respectively, the kernel size and number of the convolution layer producing low-level feature maps are 1 and 256 respectively, the kernel size and number of the convolution layer producing high-level feature maps are 3 and 128 respectively, and the hidden size of the fully connected layer is 128. For the hyperparameters in the GRU-based model, we set them as follows: the hidden size of GRU is 200, the hidden size of two fully connected layers in the attention network is 32 and 16, respectively. During training, we used the Adam optimizer with the learning rate of 0.01, and the number of epochs is 40. We set the same hyperparameters for all the systems. Also, we find that the final results are insensitive to hyperparameters within a small range. Our study is conducted on Ubuntu 16.04 with 24-core Intel Xeon E5-2690 v3 CPU(2.60GHz), 224 GB memory, 64-bit operating system, and a single NVIDIA Tesla K80 GPU accelerator.

D. Evaluation Metrics

Following the existing work [1], [16], [17], we adopted the widely-used metric, i.e., *accuracy*, to measure the effectiveness of incident-triage approaches. This metric means that the responsible team is identified as the one with the largest probability predicted by an incident triage approach among all the teams. Since the incident-triage result is refined when a new discussion item is created by engineers in the scenario of continuous incident triage, we measured accuracy when each new discussion item is created. In particular, in this scenario we hope to achieve correct triage with as few discussions as possible, and thus we measured accuracy when each of the first 5 discussion items is created.

We also measured the efficiency of each approach. For each approach, we recorded the time spent on the training stage, which is to build a classification model for incident triage, and the average time spent on the recommending stage, which is to recommend the responsible team for a new incident. For the ease of presentation, we call the former *training time* and the latter *recommendation time*.

E. Results and Analysis

[4] https://mxnet.incubator.apache.org/.

TABLE II: Statistical analysis for the accuracy among DeepCT, DL$_{no}$, and DL$_{all}$ on all the studied systems

Approach		# Discussion Items 1	2	3	4	5
Avg.	DeepCT	0.641	0.686	0.709	0.722	0.729
	DL$_{no}$	0.535	0.544	0.549	0.551	0.552
	DL$_{all}$	0.473	0.562	0.608	0.639	0.650
⇑(%)	vs DL$_{no}$	19.81	26.10	29.14	31.03	32.07
	vs DL$_{all}$	35.52	22.06	16.61	12.99	12.15
p-val	vs DL$_{no}$	**0.004**	**0.000**	**0.000**	**0.000**	**0.000**
	vs DL$_{all}$	**0.000**	**0.000**	**0.000**	**0.002**	**0.002**

1) Overall Effectiveness of DeepCT: Figure 5 shows the effectiveness of DeepCT in continuous incident triage compared with DL$_{no}$ and DL$_{all}$. Each graph in Figure 5 presents the comparison results for each studied online service system. From Figure 5, for 11 (out of 14) studied online service systems, the red line (representing the effectiveness of DeepCT) is always higher than the blue line (representing the effectiveness of DL$_{no}$) and the green line (representing the effectiveness of DL$_{all}$), when the number of discussion items increases from 1 to 5. The results demonstrate that DeepCT performs better than DL$_{no}$ and DL$_{all}$ for continuous incident triage in most cases. That DeepCT outperforming DL$_{no}$ indicates that discussions indeed provide important information for incident triage. That DeepCT outperforming DL$_{all}$ indicates that considering the characteristics of discussions (i.e., incremental creation) during learning is important for incident triage. More specifically, compared with DL$_{all}$, DeepCT enhances the learning for the knowledge from earlier discussions so that the correct assignments can be achieved as early as possible. From the results, DeepCT indeed is able to achieve correct incident-triage results with fewer discussions.

There are three systems (at the Row-2&Column-4, Row-2&Column-5, and Row-3&Column-1 in Figure 5), in which DeepCT sometimes performs worse than DL$_{no}$/DL$_{all}$, but the differences in accuracy are very small. More specifically, for the three systems, the largest difference in accuracy between DL$_{all}$ and DeepCT is only 0.018 while the largest difference in accuracy between DL$_{no}$ and DeepCT is only 0.057. Furthermore, we find that the cases DL$_{no}$ performing better than DeepCT occur when the number of discussion items is small (e.g., 1 for the system at the Row-2&Column-4), while the cases that DL$_{all}$ performing better than DeepCT occur when the number of discussion items is large (e.g., 4∼5 for the system at the Row-2&Column-5). The reason is that, when the number of discussion items is small, the title&summary has a larger impact on prediction, and thus DL$_{no}$, which ignores discussions during learning, may be more suitable at this time than later; when the number of discussion items becomes larger, the impact of discussions also increases, and thus DL$_{all}$, which treats all discussions as a whole during learning, may be more suitable at this time than before.

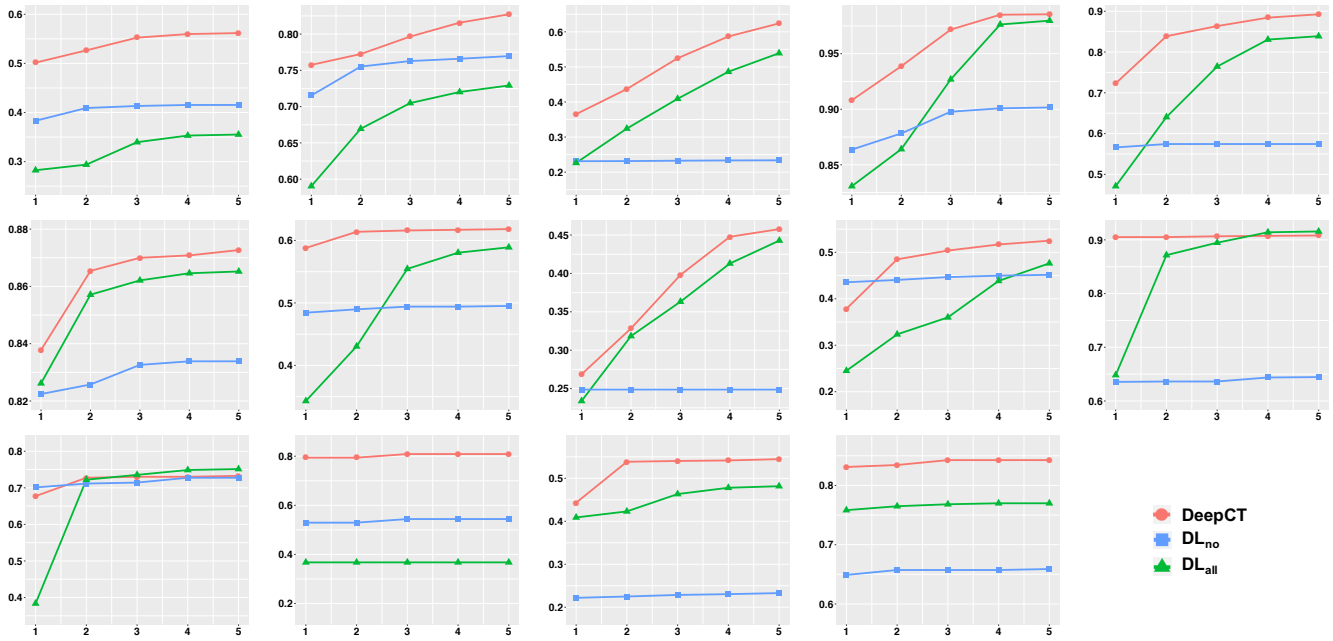

Fig. 5: Effectiveness comparison among DeepCT, DL_{no}, and DL_{all} for each studied online service system (the x-axis represents the number of discussion items and the y-axis presents the accuracy of incident triage)

We further conducted statistical analysis for the effectiveness of DeepCT on all the studied online service systems, whose results are shown in Table II. In this table, Rows 3-5 present the average accuracy on all the studied systems for DeepCT, DL_{no}, and DL_{all}, respectively. Rows 6-7 present the improved rate of accuracy for DeepCT compared with DL_{no} and DL_{all}, respectively. From these rows, the average accuracy that the responsible team is identified as the one with the largest probability predicted by DeepCT is 0.641∼0.729 with the number of discussion items increasing from 1 to 5, demonstrating the effectiveness of DeepCT. The average accuracy of DeepCT is always larger than that of DL_{no} and DL_{all} no matter how many discussion items there are. In particular, in terms of average accuracy, DeepCT improves DL_{no} by 18.92%∼30.88% and DeepCT improves DL_{all} by 12.15%∼35.52%. Moreover, we find that with the number of discussion items increasing, the improved rate compared with DL_{no} increases while the improved rate compared with DL_{all} decreases. This is as expected due to the characteristics of DL_{no} and DL_{all}. As discussed above, DL_{no} ignores discussions and thus it performs better when the number of discussion items is smaller, while DL_{all} treats all discussions as a whole and thus it performs better when the number of discussion items is larger. Here we do not compare them when there is no discussion since at that time DeepCT performs close to DL_{no}. Also, with the discussions going on (till the end), the effectiveness of DeepCT becomes closer to that of DL_{all}. That highlights that, the main contribution of DeepCT is to achieve correct incident-triage results as early as possible (i.e., with fewer discussions).

To investigate whether DeepCT significantly outperforms DL_{no} and DL_{all}, we conducted Wilcoxon signed-rank test [18] at the significant level of 0.05 between them, whose results are shown in the last two rows in Table II. Here, the bold values mean that DeepCT statistically significantly outperforms DL_{no}/DL_{all}. The results demonstrate that DeepCT indeed statistically significantly outperforms DL_{no} and DL_{all} in all the cases.

In summary, DeepCT is able to effectively relieve the problem of continuous incident triage for large-scale online service systems by achieving the average accuracy of 0.641∼0.729 with the number of discussion items increasing from 1 to 5. Moreover, DeepCT performs better than the two compared approaches (i.e., DL_{no} and DL_{all}) for most of the studied systems, and the former significantly outperforms the latter two in statistics.

2) DeepCT v.s. DeepCT$^{mask}_{no}$ v.s. DeepCT$^{loss}_{ori}$: We compared the effectiveness of DeepCT, DeepCT$^{mask}_{no}$, and DeepCT$^{loss}_{ori}$, to investigate the contributions of two main components (i.e., the attention-based mask strategy and the continuous loss function) in DeepCT, whose results are shown in Figure 6. In this figure, the box plots present the median and interquartile ranges of accuracy. The figure shows that DeepCT always performs better than both DeepCT$^{mask}_{no}$ and DeepCT$^{loss}_{ori}$ in median accuracy when the number of discussion items increases from 1 to 5. We also calculated the improved rate of average accuracy for DeepCT compared with DeepCT$^{mask}_{no}$ and DeepCT$^{loss}_{ori}$, where DeepCT improves DeepCT$^{mask}_{no}$ by 4.74%∼9.39% and improves DeepCT$^{loss}_{ori}$ by 3.70%∼36.97%. The results demonstrate that both of the

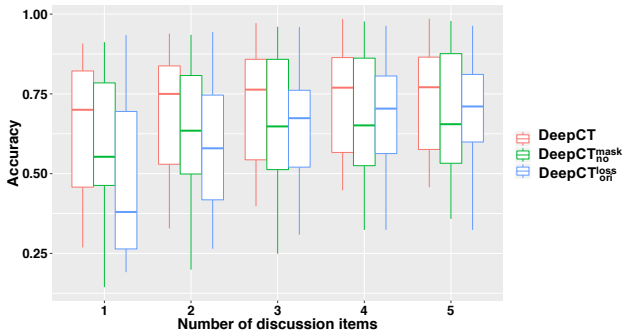

Fig. 6: Effectiveness comparison among DeepCT and its two variants on all the studied online service systems

TABLE III: Average time cost comparison among DeepCT, DL_{no}, and DL_{all} on all the studied systems

Time	**DeepCT**	**DL_{no}**	**DL_{all}**
Offline training (min)	189.07	149.99	152.80
Online recommending (ms)	0.80	1.29	1.33

components are useful and indeed contribute to DeepCT.

Furthermore, from Figure 6, when the number of discussion items is small (i.e., 1 and 2), DeepCT$^{mask}_{no}$ performs better than DeepCT$^{loss}_{ori}$ in median accuracy, while when the number of discussion items is large (i.e., 3∼5) DeepCT$^{loss}_{ori}$ performs better than DeepCT$^{mask}_{no}$. That is, the continuous loss function makes more contributions to DeepCT when the number of discussion items is small, while the attention-based mask strategy makes more contributions to DeepCT when the number of discussion items is large. The reason is that the continuous loss function aims to optimize the prediction result at each time step so that the correct assignment can be achieved as early as possible, and thus its impact is larger when the number of discussion items is smaller. With the discussions going on, its effect would be gradually closer to that of directly calculating the loss at the last time step.

3) Efficiency Comparison: We further investigated the time efficiency of DeepCT, whose results are shown in Table III. In this table, the second row presents the average training time on all the studied systems, and the last row presents the average recommending time for each new incident on all the studied systems. For all the three approaches, their average recommending time is very small (in milliseconds). That is, it is very efficient for DeepCT to conduct each triage for an incident. Besides, the average training time for the three approaches is also similar (in minutes), and that of DeepCT is slightly larger than that of the other two. However, since the training process is offline, the time cost of them is actually acceptable. Overall, DeepCT has acceptable offline training time and negligible online recommending time, indicating that DeepCT is a practical approach.

V. Lessons and Discussion

A. Lessons Learned

We summarize some of the lessons learned from our industrial experience. For Microsoft's online service systems we worked on, they are all of very large scale. They contain many sub-systems, each of which consists of many interconnected components. Each component has its own monitors that regularly check the runtime status of the component. Signals from the components reflect different aspects of system health status, such as computing resource, traffic volume, response latency, etc. Many fault-tolerant techniques (such as the "failover" mechanism[5]) are designed to ensure reliability and resiliency of the systems. Therefore, an incident to an individual component may not affect the overall system and an incident to the overall system may be reflected by many components. Although On-Call Engineers (OCEs) have rich experience and domain knowledge, they cannot fully understand the entire system and are often confused by the actual causes of an incident, resulting in incorrect initial assignments. Similarly, product teams that are responsible for maintaining individual components may not understand the details about other components and the entire system, which causes many reassignments of incidents and prolonged discussions and MTTR (the mean time from incident creation to incident resolution).

The online service systems we studied operate on 7*24 basis and receive a large number of incident reports. Our experience shows that many of these incident reports, reported by different channels or monitors, have the same root cause and are actually duplicated or linked. The ability to identify the duplicate/linked incident reports could significantly reduce the amount of discussion time. This would be investigated on our future work.

B. Generality of DeepCT

We discussed the generality of DeepCT from three aspects. First, all the 14 online service systems used in our study are in different application areas and developed by different product groups. That is, the diversity of these studied systems is large.

Second, the whole framework of DeepCT is applicable to online service systems from other companies, since the input data used in DeepCT are general and common. That is, DeepCT is easy to apply as long as there exist historical incident data for a system. In the future, we will further explore the possibility of cross-company prediction.

Third, it is also interesting to investigate whether DeepCT can be generalized to traditional software systems, besides online service systems. Here, we used the open-source traditional software system Eclipse, as the studied subject, which has been widely-used in traditional bug triage studies [19]–[22]. We used the widely-used public Eclipse dataset [23], including 47K bug reports and 564 developers. We first investigated the average number of discussion items before achieving correct assignments, and find that of Eclipse is 3.51. That

[5]https://en.wikipedia.org/wiki/Failover

indicates discussions are also conducted for refining bug-triage results in the traditional software system, but their discussions are conducted less intensively than those for online service systems due to the large scale and complexity of the latter. In particular, DeepCT achieves the accuracy (i.e., the correct developer is identified as the one with the largest predicted probability among all the 564 developers) of 0.270∼0.329 by improving 7.14%∼22.76% compared with DL_{no} and improving 2.49%∼9.31% compared with DL_{all}, when the number of discussion items increases from 1 to 3. The results demonstrate that DeepCT also performs better than the state-of-the-art bug-triage approaches (i.e., DL_{no} and DL_{all}) for the open-source traditional software in the practical scenario of continuous triage, indicating the generality of DeepCT to some degree.

C. Threats to Validity

The *internal* threat to validity mainly lies in the implementations of our approach DeepCT and compared approaches. To reduce this threat, two authors have carefully checked the code. In particular, we implemented them based on a matured framework, which has been presented in Section IV-C.

The *external* threat to validity mainly lies in the subjects. In our study, we used 14 large-scale online service systems in Microsoft. All these used data are real in industry. Even so, the used subjects may not represent the online service systems in other companies. We discussed the generality of DeepCT in Section V-B. In the future, we will apply DeepCT to more online service systems from different companies.

The *construct* threat to validity mainly lies in the used hyperparameters and metrics. To reduce the threat from hyperparameters, we tuned the hyperparameters in DeepCT and the compared approaches through grid search following the existing work [10], whose specific settings have been presented in Section IV-C. To reduce the threat from metrics, we used the most widely-used accuracy and time cost as the metrics in our study. In the future, we will use more metrics (e.g., false positive rate and re-training frequency) to more sufficiently evaluate the effectiveness and efficiency of DeepCT.

VI. Related Work

A. Incident Management

Our work aims to solve the problem of continuous incident triage in practice. Currently, there is no existing approach that is proposed to solve the same problem. The most related work to ours is the empirical study on incident triage conducted by Chen et al. [1], which investigated whether bug triage approaches for traditional software systems can effectively handle incident triage for online service systems. Their results demonstrated that traditional bug triage approaches cannot perform well for incidents, especially the incidents that are manually assigned incorrectly in the beginning. Different from their work, this work proposes the first approach to solving the problem of *continuous incident triage* for large-scale online service systems.

Besides incident triage, there are also some work on incident management. Most of research on incident management focuses on the identification of incident beacons [24]–[26], which are formed based on a combination of system metrics with unusual values produced by the incidents. For example, Cohen et al. [25] proposed a Tree-Augmented-Network (TAN) approach to deducing a TAN model, and then used the model to predict system SLO (Service Level Objective) states based on some system metrics. Here, their approach regards the system metrics used by the TAN model as service-issue beacons. Later, Cohen et al. [26] extended the work [25] by proposing a Signature approach. Besides, some work aims to associate a new incident with a previously known incident [27], [28]. For example, Duan and Babu [27] proposed an active-learning based approach to improving the overall accuracy based on both labeled and unlabeled data. It maximizes the benefits gained from newly-diagnosed unknown instances to facilitate manual labeling efforts. Furthermore, Lou et al. [29]–[31] conducted an experience report on applying software analytics to incident management of online service systems, including incident diagnosis and mitigation. Different from the above work, our work targets at incident triage, and proposes DeepCT to improve incident triage in a real-world scenario.

B. Bug Triage

In the literature, there are a lot of research on bug triage for traditional software [8], [16], [19], [32]–[48]. There are two main categories of bug-triage approaches, i.e., learning-based bug triage and information-retrieval based bug triage.

Learning-based bug triage regards the problem as a supervised classification problem, and then uses machine learning or deep learning methods to solve it. For example, Anvik et al. [49] proposed a machine-learning based bug triage approach, which first transforms the text (i.e., summary and description) in bug reports to feature vectors, and then uses Support Vector Machines (SVM) [49] to train a classifier for bug triage. Jonsson et al. [34] proposed to apply an ensemble learning method to integrate several machine learning algorithms for bug triage. Lee et al. [8] proposed a deep-learning based bug triage approach, which first converts each word in a bug report to a vector using Word2Vec [14], and then adopts Convolutional Neural Network (CNN) [15] to train a classifier for bug triage. Xuan et al. [21] proposed a machine-learning based bug triage approach by using data reduction techniques, which conducts the instance selection and feature selection for reducing training data before training a classifier.

Many bug-triage approaches are based on information-retrieval methods. For example, Naguib et al. [35] proposed to retrieve the responsible developer based on developers' expertise relevant to the topic of a bug report through Latent Dirichlet Allocation (LDA) [50]. Xia et al. [17] proposed a specialized-topic-model based approach for bug triage, which considers the product and component information of bug reports to construct the mapping between term space and topic space. Hu et al. [20] proposed to utilize historical bug-fix information for bug triage, which models the relationship between developers and source code components, and the relationship between source code components and the associated bugs.

Different from them, our work targets at incident triage for large-scale online service systems rather than bug triage for traditional software. Also, our work targets at a real-world incident-triage scenario (i.e., continuous triage), which is different from the traditional bug-triage scenario, i.e., conducting triage only once in the beginning. Although our approach is also based on deep learning, our deep neural network model is different from that used by the existing work [8].

VII. Conclusion

This paper reports our experience on incident triage for real-world, large-scale online service systems. Through an empirical study on 8 industrial online service systems, we show that in practice incident triage is a continuous process, in which engineers from different teams have to discuss intensively among themselves about an incident, and continuously refine the incident-triage result until the correct assignment is achieved. To reduce the service downtime and improve system availability, we propose **DeepCT**, a **Deep** learning based approach to automated **C**ontinuous incident **T**riage. DeepCT incorporates a novel GRU-based model with an attention-based mask strategy and a revised loss function to address the problem of continuous incident triage. Our experimental results on 14 large-scale online service systems in Microsoft show that DeepCT achieves the average accuracy of 0.641$\sim$0.729 when the number of discussion items increases from 1 to 5, and significantly outperforms the two compared approaches (i.e., DL_{no} and DL_{all}) in statistics. That demonstrates the effectiveness of DeepCT.

In the future, we will improve the proposed approach by identifying the duplicate/linked incidents. We will also explore the integration of DeepCT with service diagnosis and resolution systems to achieve more intelligent incident management.

References

[1] J. Chen, X. He, Q. Lin, Y. Xu, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "An empirical investigation of incident triage for online service systems," in *Proceedings of the 41st ACM/IEEE International Conference on Software Engineering*, 2019, to appear.

[2] Y. Chen, X. Yang, Q. Lin, H. Zhang, F. Gao, Z. Xu, Y. Dang, D. Zhang, H. Dong, Y. Xu, H. Li, and Y. Kang, "Outage prediction and diagnosis for cloud service systems," in *The World Wide Web Conference*, 2019, pp. 2659–2665.

[3] X. Zhang, Y. Xu, Q. Lin, B. Qiao, H. Zhang, Y. Dang, C. Xie, X. Yang, Q. Cheng, Z. Li, J. Chen, X. He, R. Yao, J. Lou, M. Chintalapati, F. Shen, and D. Zhang, "Robust log-based anomaly detection on unstable log data," in *Proceedings of the ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 807–817.

[4] J. Chen, J. Han, P. Sun, L. Zhang, D. Hao, and L. Zhang, "Compiler bug isolation via effective witness test program generation," in *Proceedings of the 2019 27th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2019, pp. 223–234.

[5] J. Chen, W. Hu, D. Hao, Y. Xiong, H. Zhang, L. Zhang, and B. Xie, "An empirical comparison of compiler testing techniques," in *2016 IEEE/ACM 38th International Conference on Software Engineering*, 2016, pp. 180–190.

[6] J. Chen, G. Wang, D. Hao, Y. Xiong, H. Zhang, and L. Zhang, "History-guided configuration diversification for compiler test-program generation," in *International Conference on Automated Software Engineering*, 2019, to appear.

[7] K. Cho, B. van Merrienboer, Ç. Gülçehre, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," *CoRR*, vol. abs/1406.1078, 2014.

[8] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, 2017, pp. 926–931.

[9] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "Fasttext.zip: Compressing text classification models," *arXiv preprint arXiv:1612.03651*, 2016.

[10] Y. Kim, "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*, 2014, pp. 1746–1751.

[11] R. Johnson and T. Zhang, "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, 2017, pp. 562–570.

[12] Y. Bengio, A. Courville, and P. Vincent, "Representation learning: A review and new perspectives," *IEEE transactions on pattern analysis and machine intelligence*, vol. 35, no. 8, pp. 1798–1828, 2013.

[13] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *international conference on learning representations*, 2015.

[14] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Advances in neural information processing systems*, 2013, pp. 3111–3119.

[15] C. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *Proceedings of COLING 2014, the 25th International Conference on Computational Linguistics: Technical Papers*, 2014, pp. 69–78.

[16] A. Tamrawi, T. T. Nguyen, J. M. Al-Kofahi, and T. N. Nguyen, "Fuzzy set and cache-based approach for bug triaging," in *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*, 2011, pp. 365–375.

[17] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2017.

[18] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, vol. 1, no. 6, pp. 80–83, 1945.

[19] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering*, 2009, pp. 111–120.

[20] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *IEEE 25th International Symposium on Software Reliability Engineering*, 2014, pp. 122–132.

[21] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 264–280, 2015.

[22] M. M. Rahman, G. Ruhe, and T. Zimmermann, "Optimized assignment of developers for fixing bugs an initial evaluation for eclipse projects," in *Proceedings of the Third International Symposium on Empirical Software Engineering and Measurement*, 2009, pp. 439–442.

[23] A. Lamkanfi, J. Pérez, and S. Demeyer, "The eclipse and mozilla defect tracking dataset: A genuine dataset for mining bug information," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 203–206.

[24] P. Bodik, M. Goldszmidt, A. Fox, D. B. Woodard, and H. Andersen, "Fingerprinting the datacenter: automated classification of performance crises," in *Proceedings of the 5th European conference on Computer systems*, 2010, pp. 111–124.

[25] I. Cohen, J. S. Chase, M. Goldszmidt, T. Kelly, and J. Symons, "Correlating instrumentation data to system states: A building block for automated diagnosis and control." in *OSDI*, vol. 4, 2004, pp. 16–16.

[26] I. Cohen, S. Zhang, M. Goldszmidt, J. Symons, T. Kelly, and A. Fox, "Capturing, indexing, clustering, and retrieving system history," in *ACM SIGOPS Operating Systems Review*, vol. 39, no. 5, 2005, pp. 105–118.

[27] S. Duan and S. Babu, "Guided problem diagnosis through active learning," in *International Conference on Autonomic Computing*, 2008, pp. 45–54.

[28] M. Natu, S. Patil, V. Sadaphal, and H. Vin, "Automated debugging of SLO violations in enterprise systems," in *2010 Second International Conference on Communication Systems and Networks*, 2010, pp. 1–10.

[29] J.-G. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering*, 2013, pp. 475–485.

[30] ——, "Experience report on applying software analytics in incident management of online service," *Automated Software Engineering*, vol. 24, no. 4, pp. 905–941, 2017.

[31] Q. Lin, J.-G. Lou, H. Zhang, and D. Zhang, "How to tame your online services," in *Perspectives on Data Science for Software Engineering*. Elsevier, 2016, pp. 63–65.

[32] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *28th International Conference on Software Engineering*, 2006, pp. 361–370.

[33] Y. Tian, D. Wijedasa, D. Lo, and C. Le Goues, "Learning to rank for bug report assignee recommendation," in *24th IEEE International Conference on Program Comprehension*, 2016, pp. 1–10.

[34] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1533–1578, 2016.

[35] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 22–30.

[36] Z. Lin, F. Shu, Y. Yang, C. Hu, and Q. Wang, "An empirical study on bug assignment automation using chinese bug data," in *Proceedings of the 3rd international symposium on Empirical software engineering and measurement*, 2009, pp. 451–455.

[37] G. Bortis and A. v. d. Hoek, "Porchlight: A tag-based approach to bug triaging," in *Proceedings of the 2013 International Conference on Software Engineering*, 2013, pp. 342–351.

[38] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Transactions on Software Engineering and Methodology*, vol. 20, no. 3, p. 10, 2011.

[39] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging: Leveraging q&a platforms for bug assignment," in *International Conference on Fundamental Approaches to Software Engineering*, 2016, pp. 231–248.

[40] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *IEEE International Conference on Software Maintenance*, 2010, pp. 1–10.

[41] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "A time-based approach to automatic bug report assignment," *Journal of Systems and Software*, vol. 102, pp. 109–122, 2015.

[42] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 22–30.

[43] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? simple term filtering and weighting for location-based bug report assignment recommendation," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, 2013, pp. 2–11.

[44] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proceedings of the 6th International Working Conference on Mining Software Repositories*, 2009, pp. 131–140.

[45] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *26th IEEE International Conference on Software Maintenance*, 2010, pp. 1–10.

[46] S. Wang, W. Zhang, and Q. Wang, "Fixercache: unsupervised caching active developers for diverse bug triage," in *2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement*, 2014, pp. 25:1–25:10.

[47] M. L. Vásquez, K. Hossen, H. Dang, H. H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: Bug or commit history, or code authorship?" in *28th IEEE International Conference on Software Maintenance*, 2012, pp. 451–460.

[48] A. S. Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," in *2015 IEEE International Conference on Software Maintenance and Evolution*, 2015, pp. 506–510.

[49] S. R. Gunn *et al.*, "Support vector machines for classification and regression," *ISIS technical report*, vol. 14, no. 1, pp. 5–16, 1998.

[50] M. Steyvers and T. Griffiths, "Probabilistic topic models," *Handbook of latent semantic analysis*, vol. 427, no. 7, pp. 424–440, 2007.