How Long Will it Take to Mitigate this Incident for Online Service Systems?

Weijing Wang†, Junjie Chen†, Lin Yang†, Hongyu Zhang‡, Pu Zhao§, Bo Qiao§, Yu Kang§, Qingwei Lin§
Saravanakumar Rajmohan¶, Feng Gao▲, Zhangwei Xu▲, Yingnong Dang▲, Dongmei Zhang§
†College of Intelligence and Computing, Tianjin University, Tianjin, China
‡The University of Newcastle, Callaghan, Australia
§Microsoft Research, Beijing, China; ¶Microsoft 365, Redmond, USA; ▲Microsoft Azure, Redmond, USA
{wangweijing, junjiechen, linyang}@tju.edu.cn; hongyu.zhang@newcastle.edu.au
{puzhao, boqiao, kay, qlin, saravar, fgao, zhangxu, yidang, dongmeiz}@microsoft.com

***Abstract*—Online service systems may encounter a large number of incidents, which should be mitigated as soon as possible to minimize the service disruption time and ensure high service availability. The ability to predict TTM (Time To Mitigation) of incidents can help service teams better organize the maintenance efforts. Although there are many traditional bug-fixing time prediction methods, we find that there are not readily available for incident-TTM prediction due to the characteristics of incidents. To better understand how incidents are mitigated, we conduct the first empirical study of incident TTM on 20 large-scale online service systems in Microsoft. We investigate the time distribution in the main stages of the incident life cycle and explore factors affecting TTM. Based on our empirical findings, we propose TTMPred, a deep-learning-based approach for incident-TTM prediction in a continuous triage scenario. Our model designs a two-level attention-based bidirectional GRU model to capture both the semantic information in text data and the temporal information in incremental discussions. And based on a novel continuous loss function, it builds a regression model to achieve accurate TTM prediction as much as possible at each time point of prediction. Our experiments on four large-scale online service systems in Microsoft show that TTMPred is effective and significantly outperforms the compared approaches. For example, TTMPred improves the state-of-the-art regression-based approach by 25.66% on average in terms of MAE (Mean Absolute Error).**

***Index Terms*—Incident Management, Online Service Systems, Mitigation Time, Prediction**

I. Introduction

In recent years, online service systems (such as Microsoft Azure and Office 365) have become more and more popular and millions of users around the world use such systems every day. Although dedicated efforts have been devoted to ensuring the quality of online service systems [1]–[8], incidents (i.e., unplanned interruptions and outages) are still inevitable in practice. These incidents could cause serious consequences and economic loss [9], [10]. For example, Facebook lost $89.6 million in revenue due to a service downtime that lasted for at least 14 hours on March 13, 2019[1]. Therefore, it is very important to assure the quality of online service systems and manage incidents efficiently and effectively.

To minimize service disruption time and ensure high service availability, incidents should be mitigated as soon as possible once they occur. We refer to the time required to mitigate an incident as incident TTM (Time To Mitigation). For a large and complex online service system, the number of incidents could be large and the root causes for the incidents could be different. Meanwhile, the number of operators in a service team for handling these incidents is limited. Therefore, it is challenging for a service team to handle all the incidents timely.

The ability to predict incident TTM can help effectively organize the service team's maintenance efforts. If TTM can be precisely estimated in advance, the incident-handling tasks can be better scheduled and relevant resources can be efficiently allocated. Although a number of bug-fixing time prediction approaches have been proposed for traditional software [11]–[17], they are not suitable for incident TTM. This is because the information provided by incident reports is relatively limited compared with those in bug reports. For example, due to the distributed and complex nature of online service systems, the initial incident reports do not have sufficient information that describes the abnormal behavior of the entire system. Operators need to understand more about the incidents and the responsible team through continuous discussions with the help of additional details provided by system monitors. Also, incident reports do not record enough data that some bug-fixing time prediction approaches require, such as the activity data (including code modification, code review, etc.) required by [12], [13] and the pre-defined bug category required by [11]. Therefore, existing bug-fixing time prediction approaches fail to achieve accurate results when applied to incident TTM prediction. Indeed, according to our experiment (to be presented in Section IV), the state-of-the-art bug-fixing time prediction approach, i.e., DeepLSTMPred [13], achieves only 0.5945 in F-measure on average in incident TTM prediction for four large-scale online service systems.

To better understand incident TTM, we conducted the first empirical study based on 20 large-scale online service systems in Microsoft. According to the study, we found that the time

[*]This work was done when Weijing Wang was visiting Microsoft Research. Qingwei Lin is the corresponding author for this work.

[1]https://www.ccn.com/facebooks-blackout-90-million-lost-revenue

spent on mitigating incidents after identifying the responsible team (T_3 defined in Section II) is still costly, compared with the time period from incident reporting to initial triage (T_1) and the time period spent on incident reassignment (T_2). It reveals that predicting TTM at different time points (e.g., incident reporting, initial triage, and final triage) is useful. Besides, we identified several factors affecting TTM, which can be used as features to facilitate the prediction of TTM.

Based on our empirical findings, in this paper, we propose TTMPred, the first deep-learning-based approach for TTM prediction. TTMPred considers three kinds of input data from incidents: initial incident description data (i.e., the title of an incident report), incremental discussion data (i.e., the textual conversations among operators before the time point of TTM prediction), and discrete data (such as incident severity and some environment information). TTMPred then encodes these input data to their feature representation by a two-level attention-based bidirectional GRU model. Finally, TTMPred builds a regression model to predict the specific incident TTM by designing a continuous loss function, which aims to achieve accurate prediction at each time point of TTM prediction.

To evaluate the effectiveness of TTMPred, we collected six-month incident data of four large-scale, diverse online service systems in Microsoft. The experimental results show that TTMPred is indeed effective to predict the specific incident TTM and significantly outperforms the compared approaches. For example, TTMPred improves the state-of-the-art bug-fixing time prediction approach [16] by 25.66% on average in terms of MAE (Mean Absolute Error). Since most of existing bug-fixing time prediction approaches just predict whether a bug can be fixed quickly or slowly, we also applied TTMPred to build a classification model by setting a threshold to distinguish fast and slow mitigation. Our experimental results show that TTMPred improves the state-of-the-art classification approach [13] by 19.09%~153.34% in terms of a weighted average F-measure at each time point. These results further demonstrate the effectiveness of TTMPred. Besides, our experiments confirmed the contribution of each main component in TTMPred and each kinds of input data used in TTMPred.

To sum up, this work makes the following contributions:

- We conduct the first empirical study on incident TTM based on 20 large-scale online service systems in Microsoft and obtain a series of findings.
- We propose TTMPred, the first deep-learning approach for TTM prediction at different time points.
- We conduct experiments on four large-scale online service systems in Microsoft, and the results demonstrate that TTMPred is effective and significantly outperforms the compared approaches.

II. An Empirical Study on TTM

We conducted the first empirical study to facilitate the understanding of incident TTM. In the study, we used four-year (ranging from 2017 to 2020) incident data from 20 large-scale online service systems in Microsoft. We only considered the incidents that were assigned to operators for mitigation in this study. Specifically, we first investigated the time distribution in the main stages of the incident life cycle, and then explored potential factors affecting TTM.

A. Time Distribution across Incident Life Cycle

As presented in the existing studies [2], [4], [18], the life cycle of incidents includes four main stages, i.e., incident reporting, incident triage (which refers to assigning or reassigning an incident to the responsible team/operator), incident mitigation (which refers to investigating what the problem is and mitigating it to bring the service back to normal), and incident resolution (which refers to identifying and fixing the underlying root cause of the incident through offline postmortem analysis). For a large-scale online service system, resolving an incident could take quite some time, which is unaffordable for the service, and thus minimizing TTM becomes their pursuit [1], [2], [4], [19]. In this study, we focus on the first three stages that are related to TTM. Since incident triage tends to involve multiple reassignments as reported in the existing studies [1], [4], we further split the stage of incident triage into *initial triage* and *reassignment*. Therefore, there are three time periods across the stages related to TTM, and we denote them as T_1 (the time period from incident reporting to initial triage), T_2 (the time period spent on reassignment since the initial triage), T_3 (the time period from the final assignment to incident mitigation).

First of all, we investigated the overall TTM distribution across all the study systems. We found that the distribution of TTM follows a long tail distribution, i.e., most of the incidents can be mitigated in a short time and others require much longer mitigation time. For example, the TTM of over 80% incidents is smaller than 7.55 time units and the TTM of the remaining 20% incidents ranges from 7.55 to around 802 time units[2]. Different from incidents of online services, developers of traditional shrink-wrapped software may decide to fix the bugs in the current release or defer it to a later release, which results in a lengthy bug-fixing time. According to a study on ArgoUML and PostgreSQL projects [20], the median resolution time of bugs is about 200 days. In addition, the number of incidents could be large since incidents are created as long as it triggers the condition of system monitors. Sometimes, a root cause may trigger many monitors and consequently a large number of incident reports [1]. Therefore, incident mitigation is an urgent and relatively short-lived task with large quantities. Clearly, for achieving better incident management and resource allocation, conducting accurate TTM prediction is important. In this way, operators can schedule their efforts more effectively within a short time period for better incident management.

Then, we analyzed the TTM distribution in terms of T_1, T_2, and T_3 on each studied system, in order to investigate which time period is the most costly one. Figure 1 shows the results, where x-axis represents the percentage occupied by T_1, T_2, and T_3 and y-axis represents different systems. From this figure, T_3 is the longest time period for all the studied systems, its

[2]Due to the company policy, we hide the actual time unit.

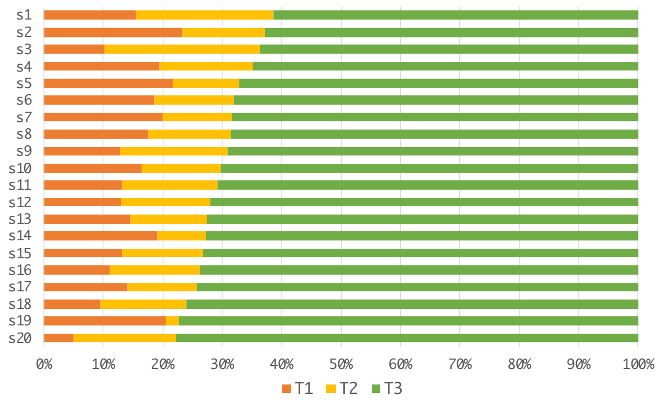


Fig. 1: Average percentage of T_1, T_2, and T_3 by different systems

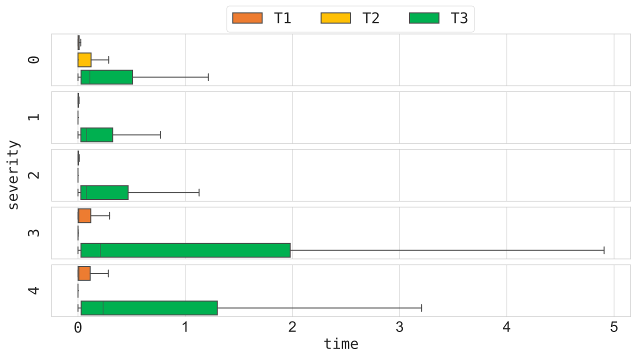


Fig. 2: TTM distribution (in terms of multiples of time) across different severity levels

percentage is ranging from 61.32% to 77.81%. The average percentage of T_1, T_2, and T_3 is 15.42%, 14.38%, and 70.20%, respectively.

This result indicates that even though the responsible team/operator for an incident is identified, mitigating the incident is still costly, and thus *predicting TTM at different time points (including the time points of incident creation, initial triage, and final triage) is meaningful.* Specifically, as time goes on (from incident creation to final triage), more information can be obtained, which is helpful to achieve more accurate TTM prediction. Here, all the time periods from these time points to incident mitigation are collectively called *TTM* for ease of presentation in this paper.

B. Influencing Factors for TTM

By carefully investigating incident reports and communicating with operators, we identified several potential factors affecting TTM, including the incident severity, the source of reporting, and the number of times of incident triage.

Incident severity is measured by the number of potentially impacted customers, which is estimated according to the region/cluster where the incident occurred. Incident severity includes five levels, i.e., 0∼4, where 0 is the highest severity and 4 is the lowest severity. Figure 2 shows the TTM distribution across different severity levels. Due to the company policy, we hide the actual time unit. We found that the incidents with higher severity levels (i.e., 0∼2) have shorter TTM than those with lower severity levels (i.e., 3∼4), indicating that incident severity is correlated to TTM to some degree. However, it alone cannot completely determine the length of TTM since the incidents with the highest severity level (i.e., 0) do not have the shortest TTM i.e., they have longer TTM than the incidents with the severity levels of 1 and 2 because of longer triage time (T_2) and longer mitigate time (T_3) as shown in Figure 2. We suspect the reason may be that these very severe incidents tend to be unusual compared with the historical incidents, so it is not easy to identify their response team and take measures to bring the problematic service back to normal based on past experience.

There are three *sources of incident reporting*, including monitor reporting, customer reporting, and engineer reporting. Figure 3 shows the TTM distribution across different incident-reporting sources. We found that the incidents reported by customers have the longest TTM while those by monitors have the shortest TTM. This is because monitors report incidents based on predefined templates and have specific location and time information, and thus they could be mitigated more quickly. Regarding customer-reporting incidents, customers tend to only describe their symptoms but not underlying information due to lack of expertise, and thus operators have to spend much time investigating these incidents. In particular, as shown in Figure 3, identifying the responsible team/operator is significantly costly compared with monitor-reporting and engineering-reporting incidents. Therefore, incident-reporting sources could be also correlated to TTM to some degree.

We also analyzed the TTM distribution across *different times of incident triage*. Intuitively, the number of times of incident

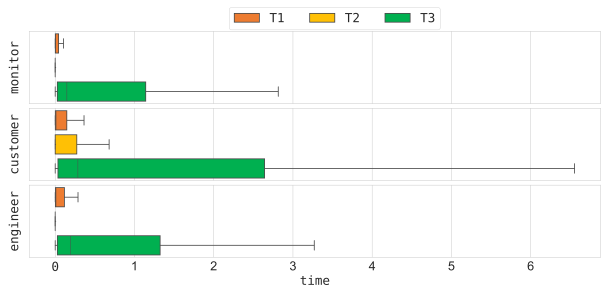


Fig. 3: TTM distribution (in terms of multiples of time) across different incident-reporting sources

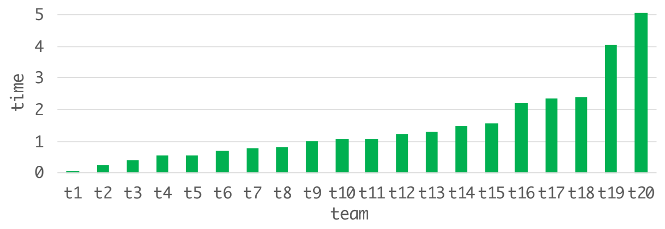


Fig. 4: TTM distribution (in terms of multiples of time) across different responsible teams

triage (i.e., the assignment/reassignment of an incident report) reflects the difficulty of diagnosing the incident and thus could be correlated to TTM. According to our investigation, as expected, the time spent on identifying the final responsible team/operator (T_2) increases obviously when the number of times of incident triage is increasing. Regarding T_3, it starts to decrease with the times of incident triage increasing from 4, since extensive discussion during incident triage provides clues for further incident mitigation. Moreover, we also found that *different responsible teams/operators* have an influence on TTM due to their different expertise and experience. Figure 4 illustrates the top 20 teams that handled most of the incidents. The result indicates that the TTM of incidents handled by different responsible teams is different. Therefore, both the number of times of incident triage and the responsible teams/operators are correlated to TTM to some degree.

C. Summary

In summary, we obtained the following major findings according to our empirical study:

- The TTM distribution on all the studied incidents has a long tail, which shows that different incidents require different TTM and a large portion of incidents have relatively short TTM.
- The time spent on mitigating incidents after identifying the responsible team/operator (T_3) is still costly (compared with T_1 and T_2), indicating that predicting TTM at a different time (including the time of incident creation, initial triage, and final triage) is meaningful.
- There are many factors affecting TTM, including incident severity, incident source, times of incident triage, and responsible teams/operators, which can all be used as features to facilitate the prediction of TTM.

III. Approach

Motivated by our empirical study described in Section II, we propose the first approach to predicting the TTM for each incident. Specifically, there are several factors that have an influence on TTM as demonstrated in the study, and thus we design a deep-learning-based approach (called TTMPred) so that these factors and text information (the core part in an incident report) can be comprehensively leveraged for TTM prediction. Since there are several kinds of input data (such as some discrete data and text data) used in TTMPred, how to effectively represent them is a major challenge in TTMPred. In particular, as demonstrated in the above study, it is meaningful to predict TTM at different time points. As time goes by, textual discussion information about the incident could be created incrementally by operators, and more sufficient information could be helpful to predict TTM. Hence, it is very necessary to handle such temporal discussion relationship created before each time point for TTM prediction, which further aggravates the challenge of representing input data.

Figure 5 shows the overview of TTMPred. For different kinds of input data (introduced in Section III-A), TTMPred designs different feature representation methods (presented in Section III-B). In particular, we design a two-level attention-based bidirectional GRU model in TTMPred in order to capture both the semantic information in text data (the first-level model, also called sentence encoder) and the temporal relationships in incremental discussion (the second-level model, also called discussion encoder). In this way, TTMPred is able to predict TTM at different time points by effectively leveraging all the input data existing at the corresponding time point (especially the incrementally provided discussion information). Based on the represented features, TTMPred builds a regression model and a classification model to predict TTM by designing a continuous loss function, which aims to fit our scenario with incremental discussion information, so that accurate prediction can be achieved as much as possible at each time point (presented in Section III-C).

A. Input Data

Inspired by our empirical study and the existing work on incidents [2], [4], [7], [19], TTMPred considers three kinds of input data as follows:

Initial text data: The title is a textual description of an incident provided when the incident report is submitted.

Incremental discussion data: After submitting an incident report, operators start to discuss this incident in order to identify what the problem is, determine which team or operator should be responsible for it, and find the strategy of mitigating it. Discussions among operators could run through the whole mitigation process (including the process of initial triage and reassignments). These discussion data are also text data related to the incident, which are incrementally written by operators like conversations. Intuitively, with discussions increasing, more sufficient information about the incident could be provided, which can further facilitate the prediction of incident TTM. At each time point for incident TTM prediction, TTMPred only considers the discussion data provided before the corresponding time point.

Discrete data: As demonstrated in our empirical study, several discrete data have an influence on TTM, and thus TTMPred considers them (i.e., incident severity, incident source, the times of incident triage, and the responsible teams and operators) as well as some other discrete data (such as the number of operators involved during the discussion process) that are recommended by some operators through face-to-face communication with them. Among these discrete data, incident severity and incident source are provided when an incident report is submitted, and the other discrete data are obtained during the process of incident triage. In particular, during the process of incident management, the severity and the responsible teams and operators may be changed with the understanding for the incident becoming deeper.

B. Feature Representation

For different kinds of input data, we design different feature representation methods in TTMPred.

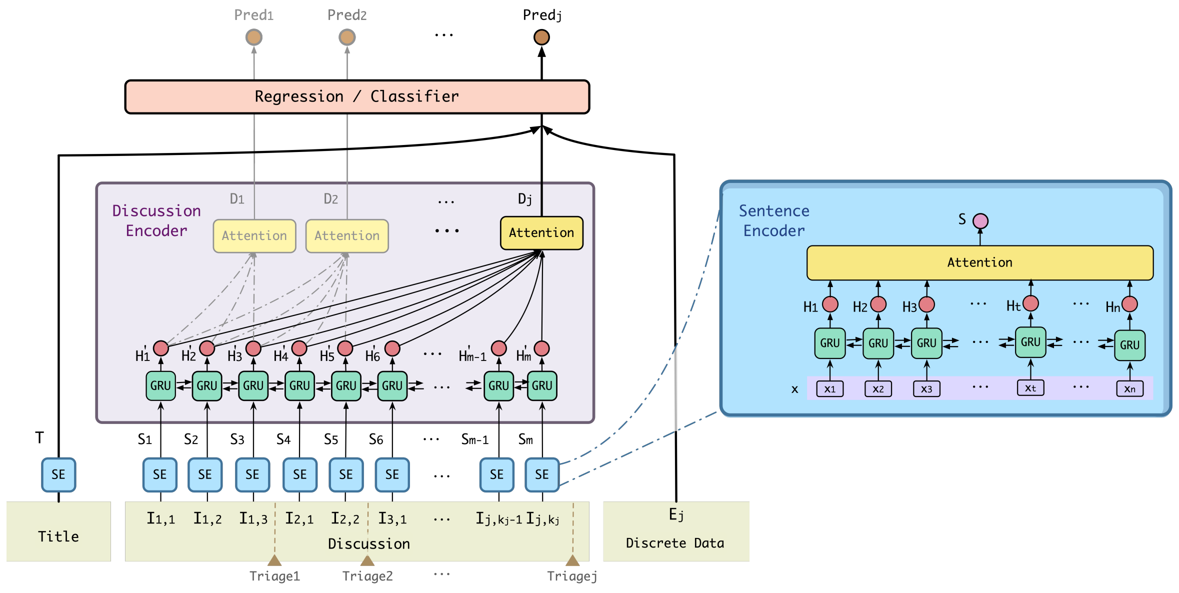


Fig. 5: Overview of TTMPred

1) Text Semantic Representation: For text data, extracting their semantic information and relieving the long discussion issue help to understand incidents better. To achieve this goal, TTMPred regards the incident title and each item of discussion as one long sentence and designs an attention-based bidirectional GRU model (also called the first-level model in this paper) as a Sentence Encoder (shown in Figure 5).

GRU [21] is a popular variant of RNN [22] and has been demonstrated to be effective and efficient in many previous studies [4], [23]. In a typical GRU, an *update gate* and a *reset gate* work together to control how much information is updated to the new state. Here, our Sentence Encoder takes a sequence of words from one sentence (i.e., each discussion item or title) as input. To obtain the vector of each word, TTMPred first builds a word-level language model through FastText [24]. We denote the vectors of a sequence of words from one sentence as $X = [x_1, x_2, \ldots, x_n]$, where x_t is the vector of the t^{th} word and also the input of the GRU at the t^{th} time step. Then, according to the previous hidden state h_{t-1}, the update gate z_t and reset gate r_t are calculated by Formula 1:

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

where σ is the logistic sigmoid function, W_z and W_r are network parameters. Afterwards, it calculates the hidden state h_t at the t^{th} time step based on h_{t-1} and x_t by Formula 2:

$$\begin{aligned} \tilde{h}_t &= tanh(W_h \cdot [r_t \odot h_{t-1}, x_t]) \\ h_t &= z_t \odot h_{t-1} + (1 - z_t) \odot \tilde{h}_t \end{aligned} \tag{2}$$

where W_h is also a network parameter and $\odot$ is an element-wise multiplication.

In particular, we design a bidirectional GRU model to learn from both directions of the input sequence, in order to capture semantic information in a complete sentence. Specifically, the forward GRU and the backward GRU calculate a hidden state at a time step respectively, and then we obtain an integrated hidden state (denoted as H) by concatenating the two hidden states from both directions.

In addition, not all the words in a sentence make the same contribution to understand the semantics, and thus we incorporate an attention mechanism to identify important words and then aggregate their representation to produce a sentence vector. Here, we denote the attention-based sentence vector integrated by $H_1, \ldots, H_n$ as S, which shows in Formula 3:

$$S = \sum_{i}^{n} \alpha_i * H_i \tag{3}$$

where α_i is the learned weight of H_i and α_i is calculated by the *softmax* function.

2) Temporal Discussion Representation: We then design the second-level attention-based bidirectional GRU model in order to capture the temporal relationships in incremental discussion and then obtain temporal discussion representation (as shown in Discussion Encoder in Figure 5). Due to the noise in operators' conversations, not all the discussion items are equally important for TTM prediction. Here, we also incorporate an attention mechanism to assign different weights to these discussion items by measuring their importance. In this way, TTMPred can predict TTM at different time points by effectively leveraging incrementally provided discussion information.

Specifically, as shown in Discussion Encoder in Figure 5, I_{j,k_j} represents the sequence of k_j discussions items between the $(j-1)^{th}$ triage and the j^{th} triage. At the j^{th} triage, the input of Discussion Encoder is a sequence of discussion vectors $[S_1, \ldots, S_m]$ (i.e., the outputs of Sentence Encoder from $I_{1,1}$ to I_{j,k_j}), while the output of Discussion Encoder is the attention-based discussion vector D_j. Here, our second-level bidirectional GRU model can incorporate the context information from neighbor discussion items into H', and the attention mechanism can help assign relatively small weights on noisy H'. Unlike the attention mechanism in Sentence Encoder, the weights are recalculated at each time point of TTM prediction (e.g., each triage) since there are incremental discussion items involved.

3) Discrete Feature Embedding: Regarding each type of discrete data, we embed it into a fixed-dimension vector and keep tuning the vector as model parameters. We denote the feature vector of the k^{th} type of discrete data as $E_{(k)} = e_{k1}, e_{k2}, ..., e_{ks}$, where s means the pre-defined dimensions for the k^{th} type of discrete data. Then, we concatenate all the feature vectors of discrete data into a vector E, where $E = E_{(1)} \oplus E_{(2)} \oplus E_{(3)} \oplus ... \oplus E_{(r)}$, where r is the number of types of discrete data and $\oplus$ refers to the concatenation operator. Since some types of discrete data may change during the triage process, we define the concatenated vector at the j^{th} time point of TTM prediction as E_j.

C. Model Building

To represent the incident at the j^{th} time point of TTM prediction, TTMPred concatenates the title representation T, the temporal discussion representation D_j, and the discrete data representation E_j as $[T; D_j; E_j]$. Then, TTMPred adopts a Multi-Layer-Perception (MLP) model to build the prediction model.

Since we aim to achieve accurate prediction as much as possible at each time point of TTM prediction, TTMPred calculates the average loss at various time points inspired by the continuous loss function proposed in the existing work [4]. As shown in Formula 4, the cumulative prediction results at each time point can be optimized when computing the gradient of the cost function during the back propagation,

$$loss = \frac{\sum_{k=1}^{n} \left[\sum_{j=1}^{t_k} loss_{k,j}\right]}{\sum_{n}^{k=1} t_k} \tag{4}$$

where n is the number of incidents in the training set, t_k refers to the number of time points for the k^{th} incident, and $loss_{k,j}$ means the loss value at the j^{th} time point for the k^{th} incident.

To predict the specific incident TTM, we propose to use $\left|A_{k,j} - F_{k,j}\right|$ as $loss_{k,j}$ to build a regression model, where $A_{k,j}$ is the actual mitigation time from the j^{th} time point to the final mitigation, while the $F_{k,j}$ is the corresponding predicted time. In addition, to compare TTMPred with existing classification-based bug-fixing time prediction approaches, we utilize the cross entropy loss as $loss_{k,j}$ to build a classification model.

IV. Evaluation

In the study, we aim to address four research questions:

- **RQ1**: How does TTMPred perform in predicting incident TTM?
- **RQ2**: Does each main component contribute to TTMPred?
- **RQ3**: Does each type of input data contribute to TTMPred?
- **RQ4**: What is the influence of main parameters in TTMPred?

A. Subjects

In the study, we used four large-scale, diverse online service systems in Microsoft as subjects, which belong to different application areas and are developed by different groups. We collected six-month incident data for each subject, and the amount of data varies largely. Since TTMPred is designed for predicting the specific time of incident mitigation, we filtered out the incidents that have not been resolved and those that were mitigated automatically. For each subject, we used the incidents from the former four months as the training set, those from the fifth month as the validation set, and the remaining one-month incidents as the testing set. In total, there are over 5GB of incident data involving 1,225 teams.

B. Compared Approaches

TTMPred is the first approach to predicting incident TTM, and thus we do not have direct compared approaches. To validate the effectiveness of existing bug-fixing time prediction approaches on TTM prediction, we adapted them to fit the incident mitigation scenario for comparison. In traditional bug-fixing time prediction, most approaches treated it as a classification problem, i.e., quick or slow fixing [11]–[13]. A few approaches [16], [25], [26] were proposed to predict specific bug-fixing time by building a regression model. Since our work aims to predict specific incident TTM, we mainly compared TTMPred with the state-of-the-art regression approach [16]. Furthermore, TTMPred is a deep-learning-based approach (including a two-level attention-based bidirectional GRU model for text semantic representation and MLP for prediction), and thus we also constructed some baselines based on traditional TF-IDF for text processing and machine learning for prediction to more sufficiently evaluate the effectiveness of TTMPred. In addition, to further investigate whether TTMPred can outperform existing classification approaches, we applied TTMPred as a classification approach by setting a threshold to distinguish quick/slow mitigation. In the following, we introduce these compared approaches in detail.

Regression. The state-of-the-art regression approach for predicting specific bug-fixing time was proposed by Ardimento et al. [16]. It first extracts important features from bug reports through Principal Component Analysis (PCA) [27], and then applies Support Vector Machine and Random Forests to build a regression model, respectively. We call them *SVR* and *RFR*.

Regarding our constructed traditional machine-learning based baselines, based on the same input data as TTMPred, we first adopted TF-IDF to represent text data (i.e., title and discussion) and then applied Linear Regression [28] (*LR*), Decision Tree Regression [29] (*DTR*), and XGBoost [30] (*XGB*) for regression respectively, instead of deep learning used in TTMPred.

Classification. We adapted three typical classification approaches in the field of traditional bug-fixing time prediction for incident TTM prediction (i.e., predicting whether an incident will be mitigated quickly or slowly):

1) Zhang et al. [11] proposed a KNN-based classification approach. It first extracts features (e.g., title and severity)

from each bug report, then applies the KNN algorithm to obtain K nearest neighbors of an incoming bug report, finally, predict whether it will be fixed quickly or slowly. We call it *KNN* in this paper.

2) Habayeb et al. [12] proposed a classification approach based on Hidden Markov Models. We call it *HMM* in this paper. *HMM* first extracts all the activities (e.g., bug triage, code changes, and severity changes) and states (e.g., "new", "assigned", "reopen", and "unconfirmed") associated with a bug. Then, it builds two Hidden Markov Models to learn the state changes based on various activities for quickly-fixed and slowly-fixed bugs, respectively. Finally, it determines whether a bug is fixed quickly or slowly by comparing the outputs of the two models.
3) Sepahvand et al. [13] proposed *DeepLSTMPred*, which first learns activity representation by the CBOW algorithm [31], then encodes an activity sequence using a LSTM model [32], finally builds a binary classifier by a fully-connected layer.

Please note that not all the features (such as activities related to code changes) used in these approaches are available in the scenario of incident TTM prediction, and thus we mapped features between the two scenarios as much as possible in order to obtain the best results of these traditional bug-fixing time prediction approaches in TTM prediction.

C. Variants of TTMPred

To answer RQ2, we constructed several variants of TTMPred to investigate the contributions of three main components in TTMPred, including the bidirectional GRU model, the attention mechanism, and the continuous loss function. The details of these variants are presented as follows.

- **TTMPred**$_{cnn}$ and **TTMPred**$_{tfidf}$ replace the first-level bidirectional GRU model in TTMPred with the TextCNN [33] model and the TF-IDF [34] algorithm, respectively. Specifically, in the TextCNN model we applied three sets of convolution kernels (i.e., 2,3,4), where each set has 100 kernels. As for the TF-IDF algorithm, we calculated the weight of each word (denoted as w) by $TF \times IDF$ and aggregated all the word vectors in a sentence by $S = \sum_{i=1}^{N} w_i \cdot x_i$, where x_i denotes the vector of the i^{th} word obtained from FastText (same as TTMPred).
- **TTMPred**$^{W}_{noAtt}$ and **TTMPred**$^{S}_{noAtt}$ remove the attention mechanism in each level of attention-based bidirectional GRU model, respectively.
- **TTMPred**$^{ori}_{loss}$ directly calculates the loss at the final time step instead of the continuous loss function used in TTMPred.

To answer RQ3, we investigated the contribution of each type of input data, including initial text data, incremental discussion data, and discrete data. Specifically, we constructed three variants i.e., **TTMPred**$^{T}_{no}$, **TTMPred**$^{D}_{no}$, and **TTMPred**$^{E}_{no}$ by removing each type of input data from TTMPred, respectively.

D. Implementations and Configurations

We implemented TTMPred based on Python 3.8 and Pytorch 1.5. Through grid search, we set the word-embedding size to 200, the size of GRU hidden states to 100, the number of GRU layers to 1, learning rate to 0.002, and epoch to 10.

Since the implementations of compared approaches are not available, we carefully re-implemented them following the descriptions in the corresponding papers. Regarding the settings of parameters in the compared approaches, we set them following the corresponding papers. If some parameter settings are not provided, we also conducted a grid search on validation sets to identify the best settings. For *HMM* [12], we used the *hmmlearn* [35] library to implement the Hidden Markov Models. For *XGB*, we used the implementation provided in the *XGBoost* python module [30]. For other machine learning methods, we adopted the implementations provided in the *scikit-learn* library [36].

We conducted all the experiments on Ubuntu 18.04.5 LTS with Intel(R) Xeon(R) CPU (2.60GHz), 64-bit operating system, and an NVIDIA Tesla P100 GRU accelerator.

E. Evaluation Metrics

In the study, we used Mean Absolute Error (MAE) to measure the effectiveness of regression approaches [37], which measures the mean of the absolute differences between the actual and predicted results by $MAE = \frac{1}{m}\sum_{i=1}^{m}|A_i - F_i|$ (A_i and F_i refer to the actual and predicted result for the i^{th} incident, respectively). The smaller MAE indicates the better regression effectiveness.

Regarding classification approaches, we used the weighted average F-measure (F) to measure their effectiveness. The F-measure for each class c_i is calculated by Formula 5,

$$F_{c_i} = \frac{2 \cdot (Precision_{c_i} \cdot Recall_{c_i})}{Precision_{c_i} + Recall_{c_i}} \tag{5}$$

and then their average F is weighted by the number of instances in each class as shown in Formula 6,

$$F = \frac{\sum_{c_i}(N_{c_i} \cdot F_{c_i})}{\sum_{c_i} N_{c_i}} \tag{6}$$

where N_{c_i} denotes that the total number of instances whose labels are c_i. The value of F is between 0 and 1, and the larger F value indicates the better classification accuracy.

F. Results and Analysis

1) RQ1: Effectiveness of TTMPred:

Regression: Table I shows the effectiveness of TTMPred compared with five baselines (i.e., the existing bug-fixing time prediction approaches *SVR* and *RFR*, and our constructed machine-learning based baselines *LR*, *DTR* and *XGB*) on four real-world online service systems (i.e., S1 to S4). We calculated the MAE results at different time points for TTM prediction, i.e., incident reporting (i.e., M_r), initial triage (i.e., M_i), and final triage (i.e., M_f), respectively. Due to the long-tail distribution, it is hard to predict TTM for the incidents mitigated slowly. Therefore, we further divided the testing data into two groups (i.e., incidents mitigated quickly and incidents

TABLE I: Experiment results of regression approaches on four online service systems

	Approach	$\mathrm{M_r}$	$\mathrm{M_r^{fast}}$	$\mathrm{M_r^{slow}}$	$\mathrm{M_i}$	$\mathrm{M_i^{fast}}$	$\mathrm{M_i^{slow}}$	$\mathrm{M_f}$	$\mathrm{M_f^{fast}}$	$\mathrm{M_f^{slow}}$
S1	**TTMPred**	**1.9864**	0.5129	2.5715	**1.7022**	0.4176	2.3311	**1.6303**	0.4336	2.2639
	SVR	2.6324	**0.5124**	3.4741	2.4144	**0.4138**	3.3938	2.3395	**0.4303**	3.3505
	RFR	2.1002	1.0561	2.5147	1.9042	1.1981	2.2499	1.8690	1.2172	2.2142
	LR	2.2785	1.2920	2.6703	2.2067	1.2127	2.6933	2.1557	1.2037	2.6597
	DTR	2.3772	1.0659	2.8978	2.1643	1.2290	2.6222	2.1304	1.2870	2.5771
	XGB	2.0032	1.1169	**2.3551**	1.8589	1.2822	**2.1412**	1.8341	1.3191	**2.1068**
S2	**TTMPred**	**2.1160**	**0.5310**	**2.6556**	**1.9323**	**0.3090**	**2.5626**	**1.9014**	**0.3175**	**2.5739**
	SVR	4.0691	0.6258	5.2475	3.6669	0.4883	4.9081	3.6113	0.5060	4.9340
	RFR	2.8197	1.2894	3.3435	2.7223	0.9527	3.4133	2.6960	0.9723	3.4302
	LR	3.5371	1.5641	4.2124	3.3192	1.4796	4.0374	3.2813	1.4363	4.0671
	DTR	2.9471	1.5904	3.4114	2.9012	0.9355	3.6687	2.8623	0.9486	3.6775
	XGB	2.7075	1.1798	3.2303	2.9025	0.9231	3.6754	2.8648	0.9395	3.6849
S3	**TTMPred**	**1.3175**	0.4904	1.9424	**1.1460**	**0.3897**	2.0471	**1.0882**	**0.3855**	2.0000
	SVR	1.3572	**0.2953**	2.1595	1.2616	0.4890	2.1823	1.2154	0.4989	2.1454
	RFR	1.5391	0.9187	2.0078	1.5495	1.1371	2.0408	1.5157	1.1341	2.0111
	LR	1.5563	1.0867	1.9111	1.3653	0.8963	**1.9241**	1.3251	0.9174	**1.8543**
	DTR	1.7494	0.8976	2.3930	1.8687	1.2152	2.6474	1.8150	1.2232	2.5831
	XGB	1.4937	1.0097	**1.8594**	1.4962	0.9954	2.0930	1.4898	1.0134	2.1079
S4	**TTMPred**	**1.1621**	0.4267	**1.8103**	**1.0380**	0.5073	1.6614	**1.0160**	0.5172	1.6229
	SVR	1.3368	**0.2611**	2.2848	1.0743	**0.2611**	2.0297	1.0466	**0.2608**	2.0026
	RFR	1.4088	0.8531	1.8986	1.2895	0.9942	1.6364	1.2733	1.0215	**1.5797**
	LR	1.5740	1.3463	1.7746	1.3596	1.1362	**1.6221**	1.3519	1.1514	1.5959
	DTR	1.4320	0.9599	1.8481	1.3348	0.8103	1.9510	1.3151	0.8208	1.9163
	XGB	1.4320	0.9599	1.8481	1.3348	0.8103	1.9510	1.3151	0.8208	1.9163
Avg.	**TTMPred**	**1.6455**	0.4903	**2.2450**	**1.4546**	**0.4059**	**2.1506**	**1.4090**	**0.4135**	**2.1152**
	SVR	2.3489	**0.4237**	3.2915	2.1043	0.4131	3.1285	2.0532	0.4240	3.1081
	RFR	1.9670	1.0293	2.4412	1.8664	1.0705	2.3351	1.8385	1.0863	2.3088
	LR	2.2365	1.3223	2.6421	2.0627	1.1812	2.5692	2.0285	1.1772	2.5443
	DTR	2.1264	1.1285	2.6376	2.0673	1.0475	2.7223	2.0307	1.0699	2.6885
	XGB	1.9091	1.0666	2.3232	1.8981	1.0028	2.4652	1.8760	1.0232	2.4540

TABLE II: Average experiment results of classification approaches on four online service systems

Approach	$0.25 \times$ median			$0.5 \times$ median			$1 \times$ median			$2 \times$ median		
	F_r	F_i	F_f	F_r	F_i	F_f	F_r	F_i	F_f	F_r	F_i	F_f
TTMPred	**0.81**	**0.78**	**0.77**	**0.79**	**0.76**	**0.76**	**0.78**	**0.76**	**0.75**	**0.77**	**0.76**	**0.76**
DeepLSTMPred	0.68	0.31	0.33	0.58	0.39	0.41	0.51	0.37	0.39	0.42	0.42	0.42
HMM	0.09	0.53	0.53	0.21	0.40	0.40	0.37	0.46	0.46	0.29	0.38	0.39
KNN	0.70	0.63	0.61	0.60	0.58	0.56	0.51	0.62	0.62	0.39	0.61	0.61

mitigated slowly) according to the median TTM on the training set, and then calculated the MAE results at different time points respectively, denoted as M_r^{fast}, M_r^{slow}, M_i^{fast}, M_i^{slow}, M_f^{fast}, and M_f^{slow}. In particular, the bold value for each metric represents the best result among all the compared approaches under the corresponding metric.

From Table I, TTMPred performs the best in terms of MAE at all the three studied time points (i.e., M_r, M_i, and M_f) for TTM prediction among all the compared approaches. The average improvement of TTMPred over *SVR*, *RFR*, *LR*, *DTR* and *XGB* across all the systems is 29.95%, 16.34%, 26.42%, 22.62% and 13.81% in terms of M_r, 30.87%, 22.06%, 29.48%, 29.63% and 23.36% in terms of M_i, and 31.38%, 23.36%, 30.54%, 30.62%, 24.89% in terms of M_f. The results demonstrate that TTMPred is effective and stable. Besides, we found that TTMPred becomes better as the time goes by (i.e., from the time point of incident reporting to the time of final triage). This is as expected since 1) more information (especially the incremental discussion information) will be leveraged by TTMPred for incident TTM prediction, and 2) the goal of our continuous loss function is to achieve accurate TTM prediction as much as possible at each time point.

In addition, TTMPred sometimes performs slightly worse than some compared regression approaches in terms of M^{fast} or M^{slow}, but actually, when these approaches perform slightly better than TTMPred in one metric, the former perform much worse than the latter in another metric. For example, SVR outperforms TTMPred by 13.58% in terms of M_r^{fast}, but performs much worse than the latter by 46.61% in terms of M_r^{slow}. The results demonstrate the overall effectiveness of TTMPred.

Classification: Table II presents the average classification results of the four studied systems under different time thresholds (i.e., 0.25, 0.5, 1 and 2 times of the median TTM). We calculated the weighted average F-measure of incidents at different time points, i.e., incident reporting (i.e., F_r), initial triage (i.e., F_i), and final triage (i.e., F_f) as shown in the header. We compared TTMPred with three existing

TABLE III: Average regression results among *TTMPred* and its variants for RQ2 on four studied systems

Approach	M_r	M_i	M_f
TTMPred	**1.6455**	**1.4546**	**1.4090**
$\mathbf{TTMPred_{cnn}}$	2.1009	1.4770	1.4335
$\mathbf{TTMPred_{tfidf}}$	2.1583	1.5303	1.4849
$\mathbf{TTMPred^{W}_{noAtt}}$	1.7104	1.5663	1.5178
$\mathbf{TTMPred^{S}_{noAtt}}$	—	1.4785	1.4331
$\mathbf{TTMPred^{ori}_{loss}}$	2.1826	1.5038	1.4610

classification-based bug-fix time prediction approaches (i.e., *DeepLSTMPred*, *HMM* and *KNN*). Please note that the bold value for each metric represents the best result among all the compared approaches.

The results from Table II demonstrate that TTMPred achieves effective and stable classification results at different time points. For example, in TTMPred, all the results in terms of F_r, F_i and F_f are larger than 0.75, and the largest difference between F_f and F_r is only 0.04. However, all the results of compared approaches at different time points do not exceed 0.70, and the results are unstable, which reflect that existing bug-fixing time prediction approaches are not suitable for incident TTM prediction.

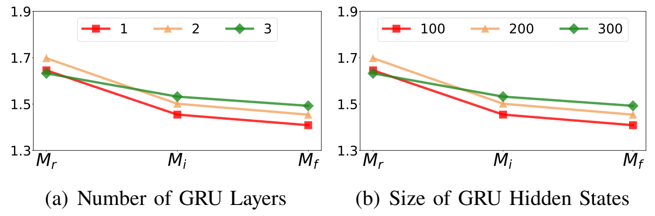
We further analyzed the reasons for these compared approaches. Regarding *DeepLSTMPred* and *HMM*, they rely on the sequences of activities and states, but these information provided by incident reports is very limited as presented in Section I. Therefore, it is difficult for them to learn useful information from the limited activities and states for incidents. Regarding *KNN*, one of the most important features used by it is the pre-defined bug category, which cannot be found in incident data, and thus the effectiveness of *KNN* is poor in the scenario of incident TTM prediction.

2) *RQ2: Contribution of each main component:* To investigate the contribution of each main component (i.e., the bidirectional GRU model, the attention mechanism, and the continuous loss function) in TTMPred, we compared TTMPred with its corresponding variants in terms of MAE at the three time points for TTM prediction. Table III shows the comparison results. By comparing TTMPred with $TTMPred_{cnn}$ and $TTMPred_{tfidf}$, we found that the former improves the latter two by 21.68% and 23.76% in terms of M_r, 1.51% and 4.94% in terms of M_i, and 1.71% and 5.11% in terms of M_f, respectively. The results demonstrate that our bidirectional GRU model is more effective to learn text semantic information for TTM prediction than TextCNN and TF-IDF.

In addition, by comparing TTMPred with $TTMPred^{W}_{noAtt}$ and $TTMPred^{S}_{noAtt}$, we found that TTMPred outperforms the latter with the average improvement of 3.79% in terms of M_r (since there is no discussion information at the time point of incident report, the second-level model does not start to contribute to TTMPred, and thus we did not evaluate $TTMPred^{S}_{noAtt}$ at this time point), 7.13% and 1.61% in terms of M_i, and 7.17% and 1.69% in terms of M_f, respectively. The results confirm the contribution of the attention mechanism in each level, which is indeed able to handle noise in text data well.

TABLE IV: Average regression results among *TTMPred* and its variants for RQ3 on four studied systems

Approach	M_r	M_i	M_f
TTMPred	**1.6455**	1.4546	1.4090
$\mathbf{TTMPred^{T}_{no}}$	2.1875	**1.4241**	**1.3774**
$\mathbf{TTMPred^{D}_{no}}$	1.6530	1.4728	1.4304
$\mathbf{TTMPred^{E}_{no}}$	1.6627	1.5233	1.4811



(a) Number of GRU Layers (b) Size of GRU Hidden States

Fig. 6: The *MAE* of various TTMPred configurations among four studied systems

By comparing TTMPred and $TTMPred^{ori}_{loss}$, we found that the former performs better than the latter with the average improvement of 24.61% in terms of M_r, 3.27% in terms of M_i, and 3.56% in terms of M_f, respectively, confirming the contribution of our continuous loss function. In particular, TTMPred (i.e., 1.6455) largely outperforms $TTMPred^{ori}_{loss}$ (i.e., 2.1826) in terms of M_r. This is because the loss function in $TTMPred^{ori}_{loss}$ only focus on the loss at the final time step, leading to worse effectiveness at the beginning.

3) *RQ3: Contribution of each type of input data:* We compared TTMPred with its corresponding three variants, whose results are shown in Table IV. By comparing TTMPred and $TTMPred^{T}_{no}$, we found that the former largely outperforms the latter with the average improvement of 24.78% in terms of M_r, but performs slightly worse than the latter in terms of M_i and M_f. This is because, at the beginning, incident title is the only text data and thus it makes very important contribution to TTMPred. With the discussion information accumulating, more accurate text data are obtained, and thus the title information may become limited, even misleading. By comparing TTMPred with $TTMPred^{D}_{no}$ and $TTMPred^{E}_{no}$, we found that the former outperforms the latter two with the average improvement of 0.45% and 1.03% in terms of M_r, 1.23% and 4.51% in terms of M_i, and 1.50% and 4.87% in terms of M_f, respectively. The results confirm the contributions of discussion data and discrete data to TTMPred.

4) *RQ4: Influence of main parameters in TTMPred:* We investigated the influence of some main parameters (i.e., the number of GRU layers and the size of GRU hidden states) in TTMPred, whose results are shown in Figure 6. In this figure, the x-axis represents three time points for TTM prediction while the y-axis represents the average MAE values on the four studied systems. We found that TTMPred is insensitive to these parameters within certain ranges.

V. Threats to Validity

The *internal* threat to validity mainly lies in the implementations of our approach and compared approaches. All the compared approaches are not open-source, thus we reimplemented them strictly following the description about them in the corresponding papers. To avoid implementation errors, we employed mature and widely-used frameworks and toolkit as presented in Section IV-D. In addition, there are many duplicate/linked incidents that have the same root cause. The relationships among incidents are marked during the incident investigation. The discussion items will only be recorded in the parent incident report, thus TTMPred may not perform well on the child incidents since they do not have complete discussion information. In the future, we will consider the linkage of incidents [3] during TTM prediction.

The *external* threat to validity mainly refers to the generalizability of our approach. First of all, the data we used in the empirical study and evaluation are from real-world online service systems in different application areas and developed by different product groups, which ensures the diversity of these systems. Even so, these systems may not represent online service systems in other companies. Due to the general framework of our approach, as long as there exists incident data with title, rich discussion, and discrete data at each time point of TTM prediction, it is easy to apply TTMPred on such datasets. In the future, we will perform a larger-scale evaluation on the systems from different companies, even on traditional software systems. Secondly, the evolution of systems may bring influence, since the knowledge learned from historical incidents may become invalid or misleading.

VI. Related Work

A. Incident management

Our work is related to incident management since predicting incident TTM at each time point can help effectively organize the service team's maintenance efforts. There are many studies on how to effectively solve the challenges posed by the incidents [1]–[4], [7], [8], [19], [38], [39]. Some studies focus on incident identification [3], [38]. For example, Lim et al. [38] proposed a Hidden Markov Random Field (HMRF) based approach to identify recurrent and unknown incidents. Furthermore, due to the complexity of online service systems, incidents could be widely propagated and cause a large number of alarms. Hence, Chen et al. [3] proposed *LiDAR* to identify linked incidents based on the relationships between components of the systems. Besides, Chen et al. [1], [4] conducted in-depth studies of incident triage in Microsoft Incident Management system, and proposed the first continuous incident triage approach *DeepCT*. Furthermore, some studies focus on accelerating the incident mitigation process from different aspects. For example, Jiang et al. [2] proposed an automated Trouble Shooting Guide recommendation approach *DeepRmd* to help operators speed up the investigation process of incidents. Chen et al. [19] prioritized incidents by predicting the probabilities of incidents being incidental to optimize incident mitigation efforts.

Different from these work, our work is the first one to predict incident TTM in incident management.

B. Bug-fixing time prediction

Our work is related to bug-fixing time prediction. Existing bug-fixing time prediction approaches can be classified into three categories: attribute-based, activity-based and text-based approaches. To construct a prediction model, researchers usually use simple partial discrete attributes (such as bug reporter, bug assignee and operating system) to construct representations of bug reports [17], [25], [40]–[45]. However, relying on discrete attributes alone does not always achieve good results. [26], [46] confirmed that some widely-used attributes do not always correlate with bug-fixing time. Besides, temporal sequences of engineers' bug-fixing activities are recorded clearly in traditional bug-fixing process, thus some approaches proposed sequential models to capture the transition information of activities, such as HMM [12] and LSTM [13]. Moreover, to extract more useful information, many studies have been devoted to explore textual description such as bug title and bug summary [11], [14]–[17], [47].

Different from the above work, our work aims at predicting the specific incident TTM value in real-word continuous triage scenario based on information that may be updated (i.e., incremental discussion and some types of discrete data), rather than predicting bug fixing-time according to the given activities sequence or the initial bug reports (i.e., without incremental discussion and updated discrete data).

VII. Conclusion

The ability to predict incident TTM (Time to Mitigation) can better schedule the mitigation efforts for better incident management. In this paper, we present the first empirical study on incident TTM based on 20 large-scale online service systems in Microsoft. During the empirical study, we investigate the time distribution in the main stages of the incident life cycle and explore potential factors affecting TTM. Based on our empirical findings, we propose TTMPred, a deep-learning-based approach for continuous TTM prediction. Our model designs a two-level attention-based bidirectional GRU model to capture text semantic information in text data and temporal information in incremental discussions. And it builds a regression model to predict the specific incident TTM by designing a continuous loss function, which aims to achieve accurate prediction as much as possible at each time point of TTM prediction. Our experiments on four large-scale diverse online service systems in Microsoft show that our TTMPred is effective and significantly outperforms the compared approaches.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant No. 62002256) and the Australian Research Council (ARC) Discovery Project (DP200102940).

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 *ICSE (SEIP)*. IEEE / ACM, 2019, pp. 111–120.

[2] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 1410–1420.

[3] Y. Chen, X. Yang, H. Dong, X. He, H. Zhang, Q. Lin, J. Chen, P. Zhao, Y. Kang, F. Gao, Z. Xu, and D. Zhang, "Identifying linked incidents in large-scale online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 304–314.

[4] J. Chen, X. He, Q. Lin, H. Zhang, D. Hao, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "Continuous incident triage for large-scale online service systems," in *ASE*. IEEE, 2019, pp. 364–375.

[5] J. Lou, Q. Lin, R. Ding, Q. Fu, D. Zhang, and T. Xie, "Software analytics for incident management of online services: An experience report," in *ASE*. IEEE, 2013, pp. 475–485.

[6] ——, "Experience report on applying software analytics in incident management of online service," *Autom. Softw. Eng.*, vol. 24, no. 4, pp. 905–941, 2017.

[7] N. Zhao, J. Chen, Z. Wang, X. Peng, G. Wang, Y. Wu, F. Zhou, Z. Feng, X. Nie, W. Zhang, K. Sui, and D. Pei, "Real-time incident prediction for online service systems," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 315–326.

[8] N. Zhao, J. Chen, X. Peng, H. Wang, X. Wu, Y. Zhang, Z. Chen, X. Zheng, X. Nie, G. Wang, Y. Wu, F. Zhou, W. Zhang, K. Sui, and D. Pei, "Understanding and handling alert storm for online service systems," in *ICSE (SEIP)*. ACM, 2020, pp. 162–171.

[9] P. Garraghan, R. Yang, Z. Wen, A. Romanovsky, J. Xu, R. Buyya, and R. Ranjan, "Emergent failures: Rethinking cloud reliability at scale," *IEEE Cloud Computing*, vol. 5, no. 5, pp. 12–21, 2018.

[10] D. Cotroneo, L. De Simone, P. Liguori, R. Natella, and N. Bidokhti, "How bad can a bug get? an empirical analysis of software failures in the openstack cloud computing platform," in *ESEC/SIGSOFT FSE*. ACM, 2019, pp. 200–211.

[11] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: an empirical study of commercial software projects," in *ICSE*. IEEE Computer Society, 2013, pp. 1042–1051.

[12] M. Habayeb, S. S. Murtaza, A. V. Miranskyy, and A. B. Bener, "On the use of hidden markov model to predict the time to fix bugs," *IEEE Trans. Software Eng.*, vol. 44, no. 12, pp. 1224–1244, 2018.

[13] R. Sepahvand, R. Akbari, and S. Hashemi, "Predicting the bug fixing time using word embedding and deep long short term memories," *IET Softw.*, vol. 14, no. 3, pp. 203–212, 2020.

[14] P. Ardimento and A. Dinapoli, "Knowledge extraction from on-line open source bug tracking systems to predict bug-fixing time," in *WIMS*. ACM, 2017, pp. 7:1–7:9.

[15] P. Ardimento, M. Bilancia, and S. Monopoli, "Predicting bug-fix time: Using standard versus topic-based text categorization techniques," in *DS*, ser. Lecture Notes in Computer Science, vol. 9956, 2016, pp. 167–182.

[16] P. Ardimento, N. Boffoli, and C. Mele, "A text-based regression approach to predict bug-fix time," in *Complex Pattern Mining*, ser. Studies in Computational Intelligence. Springer, 2020, vol. 880, pp. 63–83.

[17] C. Weiß, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *MSR*. IEEE Computer Society, 2007, p. 1.

[18] Z. Chen, Y. Kang, L. Li, X. Zhang, H. Zhang, H. Xu, Y. Zhou, L. Yang, J. Sun, Z. Xu, Y. Dang, F. Gao, P. Zhao, B. Qiao, Q. Lin, D. Zhang, and M. R. Lyu, "Towards intelligent incident management: why we need it and how we make it," in *ESEC/SIGSOFT FSE*. ACM, 2020, pp. 1487–1497.

[19] J. Chen, S. Zhang, X. He, Q. Lin, H. Zhang, D. Hao, Y. Kang, F. Gao, Z. Xu, Y. Dang, and D. Zhang, "How incidental are the incidents? characterizing and prioritizing incidents for large-scale online service systems," in *ASE*. IEEE, 2020, pp. 373–384.

[20] S. Kim and E. J. W. Jr., "How long did it take to fix bugs?" in *MSR*. ACM, 2006, pp. 173–174.

[21] 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.

[22] K. Cho, B. van Merrienboer, D. Bahdanau, and Y. Bengio, "On the properties of neural machine translation: Encoder-decoder approaches," in *SSST@EMNLP*. Association for Computational Linguistics, 2014, pp. 103–111.

[23] L. Yang, J. Chen, Z. Wang, W. Wang, J. Jiang, X. Dong, and W. Zhang, "Semi-supervised log-based anomaly detection via probabilistic label estimation," in *ICSE*. IEEE, 2021, pp. 1448–1460.

[24] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017.

[25] S. Puranik, P. Deshpande, and K. Chandrasekaran, "A novel machine learning approach for bug prediction," *Procedia Computer Science*, vol. 93, pp. 924–930, 2016.

[26] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: can we do better?" in *Proceedings of the 8th Working Conference on Mining Software Repositories*, 2011, pp. 207–210.

[27] S. Wold, K. Esbensen, and P. Geladi, "Principal component analysis," *Chemometrics and intelligent laboratory systems*, vol. 2, no. 1-3, pp. 37–52, 1987.

[28] G. A. Seber and A. J. Lee, *Linear regression analysis*. John Wiley & Sons, 2012, vol. 329.

[29] K. Osei-Bryson, "Evaluation of decision trees: a multi-criteria approach," *Comput. Oper. Res.*, vol. 31, no. 11, pp. 1933–1945, 2004.

[30] T. Chen and C. Guestrin, "Xgboost: A scalable tree boosting system," in *KDD*. ACM, 2016, pp. 785–794.

[31] T. Mikolov, I. Sutskever, K. Chen, G. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *CoRR*, vol. abs/1310.4546, 2013.

[32] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.

[33] C. N. dos Santos and M. Gatti, "Deep convolutional neural networks for sentiment analysis of short texts," in *COLING*. ACL, 2014, pp. 69–78.

[34] G. Salton and C. Buckley, "Term-weighting approaches in automatic text retrieval," *Inf. Process. Manag.*, vol. 24, no. 5, pp. 513–523, 1988.

[35] S. Lebedev, "hmmlearn," *GitHub.(https://github.com/hmmlearn/hmmlearn)*, 2016.

[36] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[37] C. Willmott, S. Ackleson, R. Davis, J. Feddema, K. Klink, D. Legates, J. O'donnell, and C. Rowe, "Statistics for the evaluation of model performance," *J. Geophys. Res*, vol. 90, no. C5, pp. 8995–9005, 1985.

[38] M. Lim, J. Lou, H. Zhang, Q. Fu, A. B. J. Teoh, Q. Lin, R. Ding, and D. Zhang, "Identifying recurrent and unknown performance issues," in *ICDM*. IEEE Computer Society, 2014, pp. 320–329.

[39] 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 *WWW*. ACM, 2019, pp. 2659–2665.

[40] E. Giger, M. Pinzger, and H. C. Gall, "Predicting the fix time of bugs," in *RSSE@ICSE*. ACM, 2010, pp. 52–56.

[41] M. Sharma, M. Kumari, and V. B. Singh, "Multi-attribute dependent bug severity and fix time prediction modeling," *Int. J. Syst. Assur. Eng. Manag.*, vol. 10, no. 5, pp. 1328–1352, 2019.

[42] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows," in *ICSE (1)*. ACM, 2010, pp. 495–504.

[43] N. Bettenburg, S. Just, A. Schröter, C. Weiss, R. Premraj, and T. Zimmermann, "What makes a good bug report?" in *SIGSOFT FSE*. ACM, 2008, pp. 308–318.

[44] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *WCRE*. IEEE Computer Society, 2012, pp. 225–234.

[45] L. D. Panjer, "Predicting eclipse bug lifetimes," in *MSR*. IEEE Computer Society, 2007, p. 29.

[46] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *PROMISE*. ACM, 2011, p. 11.

[47] R. Sawarkar, N. K. Nagwani, and S. Kumar, "Predicting bug estimation time for newly reported bug using machine learning algorithms," in *2019 IEEE 5th International Conference for Convergence in Technology (I2CT)*. IEEE, 2019, pp. 1–4.