IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 29, 2019, accepted January 6, 2020, date of publication January 10, 2020, date of current version January 17, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2965627

Continual Prediction of Bug-Fix Time Using Deep Learning-Based Activity Stream Embedding

YOUNGSEOK LEE[1], SUIN LEE[2], CHAN-GUN LEE[3], IKJUN YEOM[4], AND HONGUK WOO[4]

[1]Department of Electrical and Computer Engineering, Sungkyunkwan University, Suwon 16419, South Korea
[2]Department of Platform Software, Sungkyunkwan University, Suwon 16419, South Korea
[3]Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[4]Department of Software, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Honguk Woo (hwoo@skku.edu)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Science, ICT, and in part by the Future Planning under Grant NRF-2016R1E1A1A01943474, Grant NRF-2017R1E1A1A01075803, and Grant NRF-2018R1D1A1A02086102.

ABSTRACT Predicting the fix time of a bug is important for managing the resources and release milestones of a software development project. However, it is considered non-trivial to achieve high accuracy when predicting bug-fix times. We view that such difficulties come from the lack of continuous or posterior estimation based on subsequent developers' activities after a bug is initially reported. In this paper, we formulate the problem of bug-fix time prediction into a continual update of estimates with more activities. Logging data of bug-related activities that are streamed to a bug tracking system change the bug reports, enabling us to recalculate predictions over time. To do so, we propose a deep learning-based two-staged activity stream embedding model, DASENet that employs (i) a merged network for extracting contextual features across different types of logs, and (ii) a sequence network for exploring temporal relations of the logs. Through experiments with bug tracking system datasets from open source projects including Firefox, Chromium, and Eclipse, we show that DASENet achieves stable performance, e.g., for the Firefox dataset, top-1 accuracy of 4.6 to 8.5 % higher than other state-of-the-art works. Our approach also provides a transferable structure, yielding robust performance with a small dataset for different tasks; the DASENet model trained with a small dataset of about 900 samples (2 % of a full dataset) can show competitive performance to the other models with a full dataset. To the best of our knowledge, we are the first to employ deep learning on log streams in the context of bug-fix time prediction.

INDEX TERMS Bug-fix time, activity stream, bug tracking system, deep learning, activity embedding, sequence learning model.

I. INTRODUCTION

Data in a bug tracking system are frequently used as an essential part of managing the schedule, quality, and resources of software development in both industry practice and academic literature. Along with data, modern machine learning technology has raised the opportunity of automating several parts of bug-related processes of software development, such as bug triage [1]–[4], bug localization [5]–[11], duplicate bug detection [12]–[14], and bug priority assignment [15], [16].

One major issue for handling bugs during software development is to estimate the fix time of a bug in advance. The estimation of when a bug (or an issue) will be resolved, if sufficiently accurate, contributes to productive decision-making for resource allocation, release schedule management, and other management tasks. Several researchers investigated the problem of predicting bug-fix times, and most of this research addressed the problem by performing either regression [17], [23]–[25] or classification [18]–[22] based on the features extracted from the attributes of bug reports.

With the increasing popularity of deep learning, a few researchers have recently utilized RNNs (recurrent neural networks), CNNs (convolutional neural networks), or other neural network models to investigate features mostly available on the text of bug reports, e.g., [3], [4] for bug triage, and [7]–[11] for bug localization.

The associate editor coordinating the review of this manuscript and approving it for publication was Mervat Adib Bamiah.

In this paper, on the contrary to the purpose of those deep learning-based approaches, we focus on the continual prediction of bug-fix times and adapt deep neural networks for analyzing log streams of bug-related activities. We also use text data that can be retrieved from bug tracking systems. However, we take two different steps tailored to the contextual and temporal properties of bug-fix time prediction.

First, we consider *heterogeneity of log types*, and therefore, we develop joint learning and merged network. This network structure facilitates automated feature extraction from various types of bug-related activity logs, by combining a set of individual embedding networks wherein each is structured respectively for a specific type.

Second, we view the task of predicting the bug-fix time as *a continuous process on log streams*. Accordingly, we discuss how to continuously reevaluate the prediction with newly available data, formulating such log streams as sequences of contextual signals and then facilitating sequence analysis. This reformulation of continual prediction was motivated by our interviews with a group of project managers. They frequently talked about the benefits of automatic status updates that can not only integrate all of the ongoing activity information but also revise forecasts to the latest state, e.g., daily estimates about a set of bugs that are expected not to be resolved within a scheduled milestone.

As described above, we treat developers' activities, which are continuously logged in a bug tracking system, as signals for estimating the status of underlying bug fixing jobs. We then propose a two-staged leaning model, DASENet (Deep learning-based Activity Stream Embedding Network) that leverages the integrated use of a merged network and a sequence network; the former combines different types of logging data to a per-day activity summation, and the other generates embedding that reflects all the accumulated per-day activities in a common vector space. Through experiments with several datasets, we demonstrate that our continual prediction using DASENet is able to render stable accuracy for multi-class classification problems on bug-fix times. Our main contributions are summarized as follows:

1. We first propose a continual approach for bug-fix time prediction by exploiting deep learning techniques with data streams of bug-related activity logs. Specifically, we present the two-staged DASENet comprising a merged network and a sequence network.
2. We then validate the model with several datasets collected from the bug tracking systems of famous open source projects, i.e., Firefox, Chromium, Eclipse [26]–[28]. We reform the datasets into an easy-to-use format for sequence learning and make them available on Github [29].
3. We demonstrate the enhanced prediction ability of DASENet for bug-fix times.
4. Finally, we present a data- and time-efficient transferring procedure for variant tasks, which leverages the activity stream embedding of DASENet.

II. RELATED WORKS

In the software engineering literature, there were several works that conducted mining on bug management databases or code repositories, with the intent of achieving learning models related to bug-fix times. A summary of these works is illustrated in Table 1.

Weiss *et al*. [17] employed the text-based similarity of titles and descriptions of bug reports to predict bug fixing efforts. Zhang *et al*. [18] utilized kNN clustering techniques over bugs where the properties were characterized by domain-specific features such as the opener, priority, severity, category, and other selected bug attributes. Marks *et al*. [19] made an effort to find many attributes relevant to bug-fix times and used a decision tree with all of the attributes.

TABLE 1. Research related to bug-fix time prediction.

Researcher	*Problem Definition*	*Input data*	*Learning Model*	*Dataset (#)*	*Metric (Average %)*
Weiß [17]	Estimation of bug fixing efforts (in hours)	Textual data of tile and description fields	αkNN with Apache Lucene	JBOSS (567)	Accuracy (40) with 50% error
Zhang [18]	Bug-fix time binary classification $\begin{cases} quick : RD^a \leq \delta (= 2 \text{ to } 400 \text{ days}), \\ slow : RD > \delta \end{cases}$	Attributes (e.g., priority, severity, opener, owner)	kNN	CA Tech. (-) Unknown Project A Unknown Project B Unknown Project C	 F1 score (72) F1 score (68) F1 score (78)
Marks [19]	Bug-fix time 3-class classification $\begin{cases} 3 months : RD < 3 \text{ months}, \\ 1 year : 3 \text{ months} < RD < 1 \text{ year}, \\ 3 year : 1 \text{ year} < RD < 3 \text{ years} \end{cases}$	Attributes (e.g., bug location (product, component, version), reporter (popularity, type), description (severity, code amount))	Decision tree	Mozilla (85,616) Eclipse (63,402)	Accuracy (63) Accuracy (63)
Guo [20]	Bug classification (i.e., which bug get fixed) $\begin{cases} be \ fixed \\ not \ fixed \end{cases}$	Two attribute types; internal (e.g., severity, # of editors, reputation), external (e.g., opener and assignee in the same building ?)	Logistic regression	Windows Vista (-) Windows 7 (-)	F1 score (68) F1 score (66)
Giger [21]	Bug-fix time binary classification $\begin{cases} fast : RD \leq \delta (= 122 \text{ to } 727 \text{ days}), \\ slow : RD > \delta \end{cases}$	Attributes (e.g., priority, severity, # of CC, # of attribute changes) updated by post-submission	Decision tree	Mozilla (26,291) Eclipse (22,305) Gnome (17,063)	F1 score (68) F1 score (66) F1 score (66)
Habayeb [22]	Bug-fix time binary classification $\begin{cases} fast : RD \leq \delta (= 60 \text{ days}), \\ slow : RD > \delta \end{cases}$	Developer activity features (e.g., reporting, assignment, copy, review, file change)	HMM based on temporal sequence	Firefox (86,444)	Accuracy (71)

[a] RD denotes the remaining days until bug-fix date.

They focused on detecting outlier bugs in terms of fix time, and thus, considered loose conditions, e.g., one year-threshold.

There have been several works to exploit more than conventional bug features via handcrafted analysis on project-specific characteristics. For example, Guo *et al*. [20] particularly exploited external attributes that can be obtained from feedback of employees (not from a bug report itself). Interestingly, the examples of external attributes include geographical features such as whether the bug report opener and the assigned developer are in the same building. Giger *et al*. [21] demonstrated the benefit of exploiting post-submitted data of bug-related activities, e.g., assignee changes and developers' comments.

Recently, Habayeb *et al*. [22] exploited a hidden Markov model (HMM) to analyze temporal features of system-bug repositories, based on their sophisticated feature selections on significant events related to bug condition changes. Similar to this HMM-based approach, we concentrate on the time-dependent features of bug-related activities. However, we employ deep neural networks to incorporate diverse log types into unified feature extraction with less manual labor. Furthermore, we adopt RLSTM (residual long short-term memory)-based sequence analysis of extracted features, hence rendering our model less dependent on specific datasets.

A few researchers have demonstrated the applicability of deep learning techniques for mining bug-related data for a specific task such as bug triage and bug localization. In general, a bug triage process is to automatically find appropriate developers who can fix the bug when a bug is reported. Mani *et al*. [4] pointed out that it was critical to extract features not from individual words but from the context of sentences in bug reports in order to improve bug triage accuracy. Their work *DeepTriage* utilized a BLSTM (bi-direction long short-term memory) network where word-embedding sequences for bug description texts are fed. Similarly, Lee *et al*. [3] utilized CNNs for bug triage.

For bug localization, Lam *et al*. [7], [8] proposed a deep learning-based method using the combination of three types of features: textual similarity, relevancy, and statistical data. Specifically, they extracted the features by using a revised Vector Space Model [30], [31] with textual similarity measurements, and then derived the text and code-level relevancy between bug reports and source files. Xiao *et al*. [9]–[11] presented a CNN-based merged network structure, namely *DeepLoc*, for combining different feature types of bug reports and source files. They achieved stable performance with average accuracy 59 to 70 % and 69 to 79 % respectively for top-5 and top-10 rankings. This work and our model share a common structure in that both leverage the merged deep learning structure. However, our problem domain is not bug localization but bug-fix time prediction, which is suited to continual prediction with log streams.

At this time, not many deep learning-based methods have been introduced to solve problems related to bug-fix times. To the best of our knowledge, in the context of learning temporal properties of bug-related data, our work is the first that utilizes deep learning and embedding techniques.

III. PROPOSED APPROACH

In this section, we briefly describe our view on data in bug reports, and present the overall structure of our proposed DASENet model.

A. ACTIVITY LOG STREAMS

A bug tracking system manages databases of bug-related information and allows a group of developers in a project team to manage various issues of bugs. For each bug, its report contains relevant information such as title, owner, status, description, comment, and attachment. While there might be several different report formats and layouts across projects, it is commonly observed that a bug report is continuously updated during its lifetime alongside bug-related activities. Activities are normally logged, either by manual descriptive text of developers in natural language (e.g., description, comments, and inquires) or by semi-automated records in structured text (e.g., appending a new developer to CC list, and updating the status field from NULL to Fixed). Considering the different text structure and input source, throughout this paper, we distinguish these two appended log types into *user comment* and *system record* respectively.

Fig. 1 illustrates an example bug report in the Chromium project repository [27]. In the figure, a bug was initially reported on March 10, 2014, and then four user comments (ⓐ, ⓒ, ⓕ, ⓖ) and three system records (ⓑ, ⓓ, ⓔ) were appended until the bug status was finally changed to ''fixed'' on March 26, 2014. *User comments* contain sentences, e.g., ''Either solution is better than . . .'' and ''Change the streamPrivate . . .'', while *system records* briefly describe the changes of bug report attributes, e.g., ''CC: darine@chromium.org'' and ''Labels: Cr-Internals. . .''.

In the same vein of previous works [18]–[22], we consider day-granularity for bug-fix times. Thus, we devise a per-day bin container (namely, *activity bin*). Specifically, for each bug, all of the temporally co-logged activity information during a day is bundled together. Furthermore, we convert a set of continuously generated activity bins to the chronologically ordered sequence format (namely, *bin-sequence*).

B. OVERALL MODEL ARCHITECTURE

To achieve robust prediction models over log streams, we investigate a deep learning-based embedding structure in which both *contextual* and *temporal* features of activity log streams with various log types can be well-represented. Based on the embedding structure, we then explore various bug-related temporal tasks including our main task, bug-fix time prediction. Several tasks will be discussed in Section VI.

Fig. 2 illustrates our proposed DASENet model. Its learning process takes the following steps.

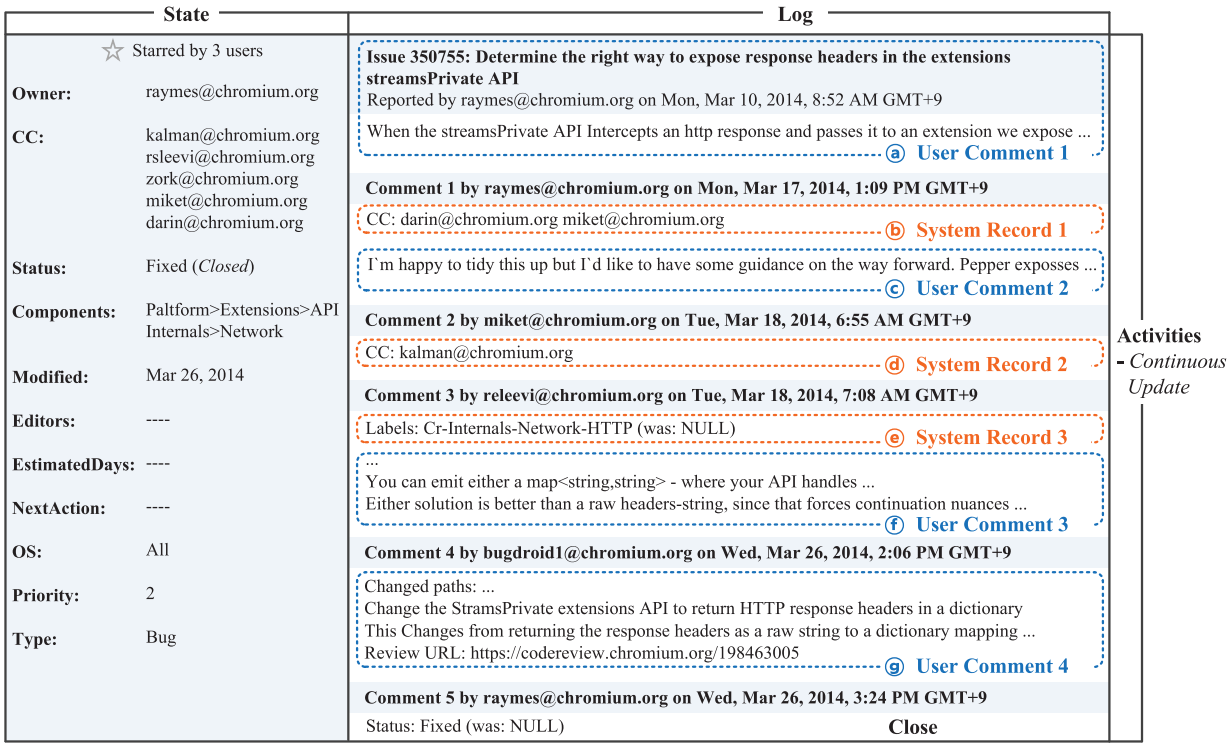


FIGURE 1. A bug report example with activity logs in the Chromium project repository.

① In *data preprocessing*, each bug report is transformed into a chronologically ordered set of per-day activity bins (bin-sequence) containing user comments, system records, and metadata.

② For *activity bin embedding*, each activity bin is used to train a *merged* network of BLSTMs and MLPs (multilayer perceptrons). This embedding process is intended for extracting contextual features related to bug fixing jobs available logging data of different types.

③ For *bin-sequence embedding*, an RLSTM-based *sequence* network is stacked on top of the merged network (in ②). The bin-sequence embedding is intended for capturing temporal relations of activities. A bin-sequence is used by such a two-staged embedding model of the merged and sequence networks, namely DASENet.

④ For *task learning*, a feed-forward MLP (task layer) is additionally stacked on the pre-trained DASENet. The task layer is directly connected with the output of the sequence network, and then separately trained with datasets for a specific task.

The implementation and use of DASENet will be discussed in the next sections: ① preprocessing in Section IV, ② activity bin embedding and ③ bin-sequence embedding in Section V, and ④ task learning on DASENet in Section VI.

Here, we briefly explain our motivation for the two-staged deep learning structure of DASENet. Table 2 shows the accuracy result of our initial test for a bug-fix time classification problem with three classes. For this test, we implemented an MLP-based deep neural network. Considering divergent perspectives on how to interpret log streams, we intentionally created three different datasets, and then trained models with each dataset. Fig. 3 depicts the difference between the datasets: (1) "cumulative - all" contains all the available logging data from the bug-open to a specific point in time for prediction including the both log types, user comments and system records for each bug report, (2) "cumulative - user comment" consists of all the available user comments only (no system records), and (3) "latest - all" holds only the latest updated per-day activity for the both log types.

In this test, we expected that a model trained with more data in terms of aggregation and log types (i.e., "cumulative - all" dataset) should outperform the other models. However, in Table 2, we observed only slight differences of 1 to 2 % in model accuracy that rarely met our expectation. This initial result led us to devise a hierarchical model structure that can extract both contextual and temporal features from time-series log streams of different types.

TABLE 2. Initial top-1 accuracy of 3-class classification.

Aggregation Type - Log Type	*Firefox*	*Chromium*	*Eclipse*
Cumulative - All	51.5	48.9	48.9
Cumulative - User Comment	50.1	48.8	45.9
Latest Only - All	50.6	47.5	46.2

Note that the datasets used here will be explained in Section VII-A. In addition, the MLP-based deep neural

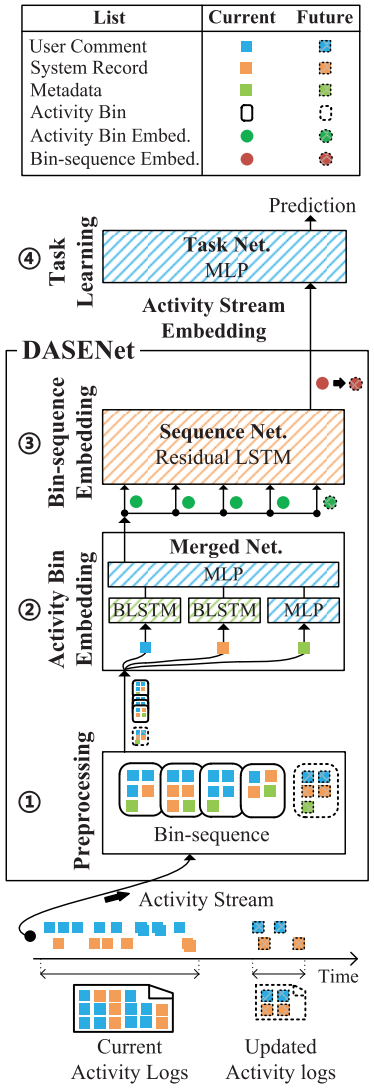


FIGURE 2. Our overall approach.

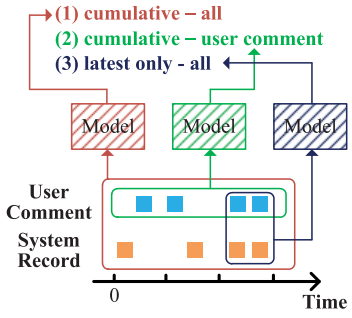


FIGURE 3. Aggregation and log types.

network model with the "cumulative - all" dataset is used as a baseline for evaluation (*DeepBase* in Section VII-B).

IV. STREAM DATA PREPROCESSING

In this section, we present how to transform log streams of various types of bug-related activities into input data for deep learning models.

TABLE 3. Metadata of activity bins.

Metadata	*Definition*
$\mathbf{m}_1$	Elapsed time from bug open date
$\mathbf{m}_2$	Existence of activities in recent 3 days
$\mathbf{m}_3$	# of unique comment authors
$\mathbf{m}_4$	Type of the latest comment author
$\mathbf{m}_5$	Cumulative # of activity logs

As described previously, a per-day activity bin B^i contains all of the co-logged activity information in various types (e.g., user comments, system records) during the same day. Alongside log types of user comments $\mathbf{X}_{\text{user}}$ and system records $\mathbf{X}_{\text{sys}}$, we also treat several statistical properties as a *metadata* set (in Table 3) for each activity bin, and include the metadata set in its associated activity bin. Then, an **activity bin** is denoted by a set

$$B^i = \{\mathbf{X}^i_{\text{user}},\ \mathbf{X}^i_{\text{sys}},\ \mathbf{X}^i_{\text{meta}}\} \tag{1}$$

where a *bin index* $i \in \mathbb{N}$ is given in chronological order.

Fig. 4 shows the timeline of individual activity logs extracted from the bug report R in Fig. 1. For example, the activity bin B^2 on March 17, 2014, holds a user comment ⓒ, a system record ⓑ, and its metadata. It should be noted that any logs trivially signifying that a bug has been closed can be seen as a label for bug-fix time prediction; hence, those should not be in activity bins. In Fig. 1 and 4, the status change log (*Fixed*) on March 26, 2014, is not included in any activity bin.

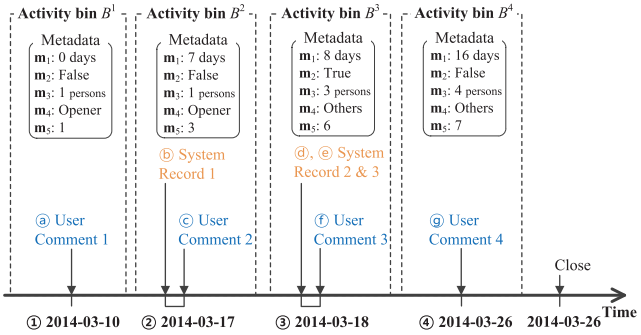


FIGURE 4. Activity timeline of a bug report.

Word embedding schemes (i.e., word2vec [32]) are used to represent individual words in user comments and system records in a common vector space. Accordingly, we denote them as vector sequences

$$\mathbf{X}^i_{\text{user}} = [\mathbf{v}_1, \ldots, \mathbf{v}_{l_{\text{user}}}], \quad \mathbf{X}^i_{\text{sys}} = [\mathbf{v}_1, \ldots, \mathbf{v}_{l_{\text{sys}}}] \tag{2}$$

where $\mathbf{v}$ is a word embedding and l_{user} and l_{sys} are the sequence lengths of $\mathbf{X}^i_{\text{user}}$ and $\mathbf{X}^i_{\text{sys}}$ respectively. In addition, we represent the metadata set in Table 3 as a binary encoded data

$$\mathbf{X}_{\text{meta}} = [\mathbf{m}_1, \ldots, \mathbf{m}_{l_{\text{meta}}}] \tag{3}$$

where $l_{\text{meta}} = 5$ is the size of $\mathbf{X}_{\text{meta}}$.

For a bug report R, its **bin-sequence** contains activity bins in chronological order, and it is denoted as

$$S^T = \{B^1, B^2, ..., B^T\} \tag{4}$$

where $T \in \mathbb{N}$ is the latest bin index at the current time.

V. ACTIVITY STREAM EMBEDDING

In this section, we present our deep learning-based activity stream embedding technique. Our proposed two-staged learning DASENet employs (i) a merged network structure with different neural networks for performing compositional embedding of various log types, and (ii) an RLSTM-based sequence network for performing temporal data analysis over log streams. These merged and sequence networks together are intended for continuously extracting contextual and temporal features from log streams in heterogeneous formats over time.

A. NETWORK COMPONENTS

DASENet is organized into deep neural network components to be laid out according to the characteristics of the input data, as shown in Fig. 5. We explain each network block such as deep BLSTM [33], MLP, and RLSTM, and depict their internal structure on the right side of Fig. 5.

1) DEEP BLSTM

Consider an l-length input vector sequence $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_l]$. A deep BLSTM function with d-hidden layers computes an output vector:

$$\mathbf{y} = [y_1, \ldots, y_o] = \mathsf{BLSTM}([\mathbf{x}_1, \ldots, \mathbf{x}_l]). \tag{5}$$

To do so, it calculates the intermediate output vector sequence $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$ and the hidden state vector sequence of the nth layer $\mathbf{H}^n = \left[\mathbf{h}_1^n, \ldots, \mathbf{h}_l^n\right]$ by iterating the equations below for $n = 1, \ldots, d$ and $t = 1, \ldots, l$. Here, we denote a forward layer as $\overrightarrow{\cdot}$, and a backward layer as $\overleftarrow{\cdot}$ for bi-directional processing.

$$\overrightarrow{\mathbf{h}}_t^n = \mathcal{H}\left(W_{\overrightarrow{\mathbf{h}}^{n-1}\overrightarrow{\mathbf{h}}^n}\overrightarrow{\mathbf{h}}_t^{n-1} + W_{\overrightarrow{\mathbf{h}}^n\overrightarrow{\mathbf{h}}^n}\overrightarrow{\mathbf{h}}_{t-1}^n + \overrightarrow{\mathbf{b}}_t^n\right) \tag{6}$$

$$\overleftarrow{\mathbf{h}}_t^n = \mathcal{H}\left(W_{\overleftarrow{\mathbf{h}}^{n-1}\overleftarrow{\mathbf{h}}^n}\overleftarrow{\mathbf{h}}_t^{n-1} + W_{\overleftarrow{\mathbf{h}}^n\overleftarrow{\mathbf{h}}^n}\overleftarrow{\mathbf{h}}_{t-1}^n + \overleftarrow{\mathbf{b}}_t^n\right) \tag{7}$$

$$\overrightarrow{\mathbf{y}}_t = W_{\overrightarrow{\mathbf{h}}^d\overrightarrow{\mathbf{y}}}\overrightarrow{\mathbf{h}}_t^d + W_{\overrightarrow{\mathbf{h}}^d\overrightarrow{\mathbf{y}}}\overrightarrow{\mathbf{h}}_t^d + \overrightarrow{\mathbf{b}}_t^{\mathbf{y}} \tag{8}$$

$$\overleftarrow{\mathbf{y}}_t = W_{\overleftarrow{\mathbf{h}}^d\overleftarrow{\mathbf{y}}}\overleftarrow{\mathbf{h}}_t^d + W_{\overleftarrow{\mathbf{h}}^d\overleftarrow{\mathbf{y}}}\overleftarrow{\mathbf{h}}_t^d + \overleftarrow{\mathbf{b}}_t^{\mathbf{y}} \tag{9}$$

$\mathcal{H}(\cdot)$ is the hidden layer function which uses memory cells defined in [34], [35]. $W_{\mathbf{h}^{n-1}\mathbf{h}^n}$ is the hidden weight matrix from the $(n-1)$th layer to the nth layer, $W_{\mathbf{h}^n\mathbf{h}^n}$ is the recurrent hidden weight matrix at the nth layer, and $W_{\mathbf{h}^d\mathbf{y}}$ is the dth hidden-output weight matrix. $\mathbf{b}$ is the bias vector. Then, the output vector is derived from the intermediate output vector sequence $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$ by:

$$\mathbf{y} = \left[\left(\overrightarrow{\mathbf{y}}_l\right), \quad \left(\overleftarrow{\mathbf{y}}_1\right)\right]. \tag{10}$$

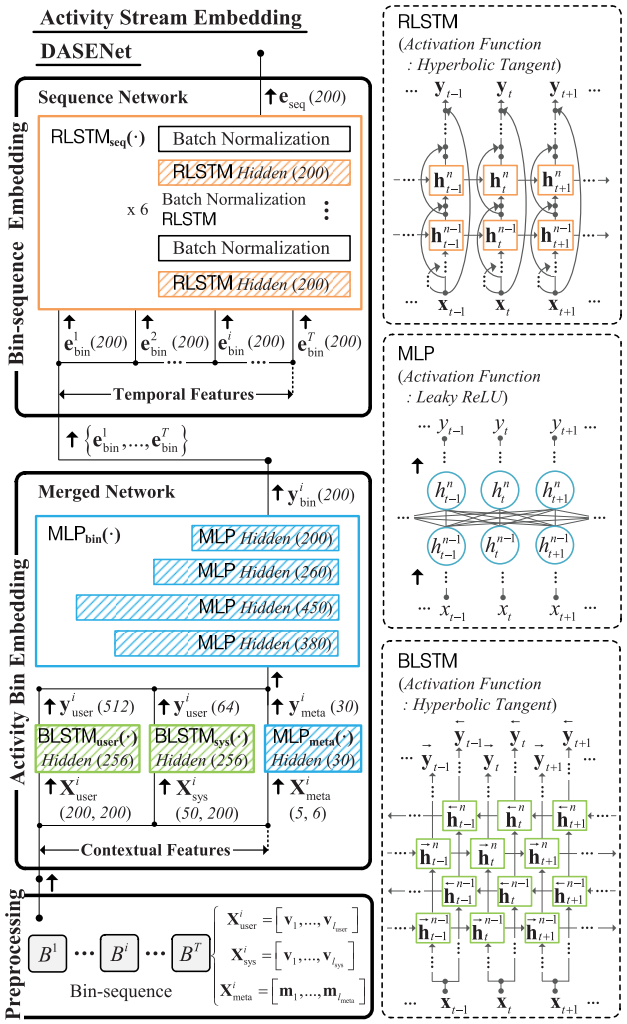


FIGURE 5. Architecture of DASENet.

2) DEEP MLP

For an input vector $\mathbf{x} = [x_1, \ldots, x_l]$, a deep MLP function yields the output vector

$$\mathbf{y} = [y_1, \ldots, y_o] = \mathsf{MLP}([x_1, \ldots, x_l]) \tag{11}$$

by calculating the hidden state vector of the nth layer $\mathbf{h}^n = [h_1^n, \ldots, h_{q^n}^n]$ where q^n is the number of the nodes in the nth hidden layer. This calculation is done by the following equations for $n = 1, \ldots, d$ and $t = 1, \ldots, q^n$:

$$h_t^n = f\left(\sum_{j=1}^{q^{n-1}} w_{h_j^{n-1}h_t^n} h_j^{n-1} + b_t^n\right) \tag{12}$$

$$y_p = f\left(\sum_{j=1}^{q^d} w_{h_j^d y_p} h_j^d + b_p^y\right), \quad p = 1, \ldots, o. \tag{13}$$

Note that $w_{h_j^{n-1}h_t^n}$ is the weight value from the jth node of the $(n-1)$th hidden layer to the tth node of the nth hidden layer. b is the bias value (b_p^y is the bias value in the pth node of the output layer), and $f(\cdot)$ is an activation function,

e.g., Leaky ReLU. In addition, $w_{h_j^d y_p}$ is the weight value from the jth node in the final hidden layer to the pth node in the output layer.

3) RESIDUAL LSTM (RLSTM)

We devise a modified LSTM network based on the residual network structure [36], which can enrich temporal feature extraction and mitigate the accuracy saturation and degradation problem of deeply layered networks [37], [38].

For an l-length input vector sequence $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_l]$, an RLSTM function computes an output vector:

$$\mathbf{y} = [y_1, \ldots, y_o] = \mathsf{RLSTM}([\mathbf{x}_1, \ldots, \mathbf{x}_l]). \tag{14}$$

To do so, it calculates the intermediate output vector sequence $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$, and the hidden state vector sequence of each layer n, $\mathbf{H}^n = \left[\mathbf{h}_1^n, \ldots, \mathbf{h}_l^n\right]$ by iterating the following equations for $n = 1, \ldots, d$ and $t = 1, \ldots, l$:

$$\mathbf{h}_t^n = \mathcal{H}\left(W_{\mathbf{h}^{n-1}\mathbf{h}^n}\mathbf{h}_t^{n-1} + W_{\mathbf{h}^n\mathbf{h}^n}\mathbf{h}_{t-1}^n + \mathbf{b}_t^{n-1}\right) \tag{15}$$

$$\mathbf{H}^n = \mathbf{H}^n + \mathbf{H}^{n-1}, \quad \mathbf{y}_t = W_{\mathbf{h}^d\mathbf{y}}\mathbf{h}_t^d + \mathbf{b}_t^{\mathbf{y}} + \mathbf{x}_t. \tag{16}$$

Then, an output vector $\mathbf{y}$ is obtained from the intermediate output vector sequence $\mathbf{Y} = [\mathbf{y}_1, \ldots, \mathbf{y}_l]$ by $\mathbf{y} = \mathbf{y}_l$ where $\mathbf{y}_l$ is the last element of $\mathbf{Y}$.

B. ACTIVITY BIN EMBEDDING

In DASENet, a merged network for activity bin embedding is jointly trained to combine various types of activity logs via multiple deep neural networks. It was pointed out that a deep neural network structure with multiple networks is effective in representing an event in the context of event embedding [39], [40]. The merged network of DASENet adapts this event embedding structure for activity log streams. Specifically, it combines deep BLSTM and MLP networks, and transforms an activity bin $B^i = \{\mathbf{X}^i_{\text{user}}, \mathbf{X}^i_{\text{sys}}, \mathbf{X}^i_{\text{meta}}\}$ (in (1)) into a bin embedding vector $\mathbf{e}^i_{\text{bin}}$.

For $\mathbf{X}^i_{\text{user}}$ and $\mathbf{X}^i_{\text{sys}}$ (in (2)) in a word embedding sequence, their feature vectors $\mathbf{y}^i_{\text{user}} \in \mathbb{R}^{o_{\text{user}}}$ and $\mathbf{y}^i_{\text{sys}} \in \mathbb{R}^{o_{\text{sys}}}$ are calculated by the deep BLSTM functions that discover underlying information from written texts:

$$\mathbf{y}^i_{\text{user}} = \mathsf{BLSTM}_{\text{user}}(\mathbf{X}^i_{\text{user}}), \quad \mathbf{y}^i_{\text{sys}} = \mathsf{BLSTM}_{\text{sys}}(\mathbf{X}^i_{\text{sys}}). \tag{17}$$

Notice that the $\mathsf{BLSTM}_{\text{user}}(\cdot)$ and $\mathsf{BLSTM}_{\text{sys}}(\cdot)$ functions do not share their parameters. The feature vector $\mathbf{y}^i_{\text{meta}} \in \mathbb{R}^{o_{\text{meta}}}$ of metadata $\mathbf{X}^i_{\text{meta}} = [\mathbf{m}_1, \ldots, \mathbf{m}_{l_{\text{meta}}}]$ (in (3)) is extracted by the MLP function:

$$\mathbf{y}^i_{\text{meta}} = \mathsf{MLP}_{\text{meta}}\left([(\mathbf{m}_1), \ldots, (\mathbf{m}_{l_{\text{meta}}})]\right). \tag{18}$$

Subsequently, the three outputs separately computed in (17)-(18) are concatenated and then fed to the deep MLP function to be merged:

$$\mathbf{e}^i_{\text{bin}} = \mathsf{MLP}_{\text{bin}}\left(\left[\left(\mathbf{y}^i_{\text{user}}\right), \left(\mathbf{y}^i_{\text{sys}}\right), \left(\mathbf{y}^i_{\text{meta}}\right)\right]\right). \tag{19}$$

Then, we treat the output $\mathbf{e}^i_{\text{bin}} \in \mathbb{R}^{o_{\text{user}}+o_{\text{sys}}+o_{\text{meta}}}$ as an **activity bin embedding** vector for the activity bin B^i. In overall, this embedding vector represents comprehensive features of B^i on a continuous vector space, combining different types of logs in B^i.

C. BIN-SEQUENCE EMBEDDING

In DASENet, a sequence network is stacked on the previously explained merged network. It takes as an input a set of activity bin embedding vectors $\left\{\mathbf{e}^i_{\text{bin}}\right\}_{i=1}^T$ (in (19)) where T denotes the number of activity bins currently available for a bug report, and then it produces a **bin-sequence embedding** vector $\mathbf{e}_{\text{seq}}$.

Recall that $\left\{\mathbf{e}^i_{\text{bin}}\right\}_{i=1}^T$ is calculated by the iteration of (17)-(19) for each of all the activity bins in S^T (in (4)). Then, the deep RLSTM function computes its embedding vector:

$$\mathbf{e}_{\text{seq}} = \mathsf{RLSTM}_{\text{seq}}\left(\left\{\mathbf{e}^i_{\text{bin}}\right\}_{i=1}^T\right). \tag{20}$$

This embedding vector $\mathbf{e}_{\text{seq}}$ turns out to encapsulate both contextual and temporal features of activity log streams via the stacked networks of bin embedding and bin-sequence embedding.

D. DASENET MODEL IMPLEMENTATION

Fig. 5 illustrates the DASENet implementation, where the dimensions of input and output variables and the number of hidden nodes are presented in parentheses. For example, with a word being mapped to 200-dimensional data, $\mathbf{X}^i_{\text{sys}}(50, 200)$ in the box of the activity bin embedding indicates a 50-length sequence input for a system record (in (2)). In addition, $\mathbf{e}_{\text{seq}}(200)$ on the top of the figure denotes that the size of our bin-sequence embedding vector (in (20)) is 200.

To train DASENet, we leverage the notion of supervised embedding [41] that is known to be effective for a specific task or a group of transferable tasks, and particularly, we employ the bug-fix time prediction task with K-classes. Specifically, we stack an additional layer of $\mathsf{MLP}_{\text{train}}(\cdot)$ and a softmax function on top of DASENet. $\mathsf{MLP}_{\text{train}}(\cdot)$ takes the output $\mathbf{e}_{\text{seq}}$ of DASENet and yields a K-size vector $\mathbf{y}_{\text{train}}$.

$$\mathbf{y}_{\text{train}} = [y_1, \ldots, y_K] = \mathsf{MLP}_{\text{train}}(\mathbf{e}_{\text{seq}}) \tag{21}$$

Then, the predicted probability of jth element (y_j) in $\mathbf{y}_{\text{train}}$ is calculated by a softmax function:

$$\mathsf{Pr}_{\text{softmax}}(j) = \frac{\exp\left(y_j\right)}{\sum_{p=1}^K \exp\left(y_p\right)}. \tag{22}$$

For training DASENet in supervised manner, we utilize categorical cross-entropy errors of K-class bug-fix time classification:

$$\text{Cross-entropy error} = -\sum_{j=1}^{K} c_j \log\left(\mathsf{Pr}_{\text{softmax}}(j)\right) \tag{23}$$

where c_j is ground truth of the class label j.

Specifically, we set $K = 7$, by considering that the features of embedding outputs can be biased unless the distribution of

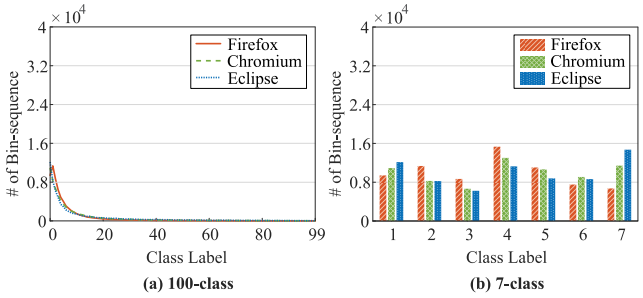


FIGURE 6. Distribution of bin-sequences.

samples is well-balanced. Fig. 6 compares the distributions of bin-sequence samples; in Fig. 6(b), we notice that the distribution of 7-class is relatively balanced, as contrasted to imbalanced long-tail pattern of 100-class in Fig. 6(a). Note that the criterion of 7-class is in Table 4.

TABLE 4. Multiple class ranges for three tasks.

Task	*#-class*	*Class Value Intervals*
FixTime	2-class	{ [0 3], (3 100] } for Firefox { [0 4], (4 100] } for Chromium { [0 5], (5 100] } for Eclipse
	3-class	{ [0 1], (1 5], (5 100] }
	5-class	{ [0 0], (0 2], (2 5], (5 20], (20 100] }
	7-class	{ [0 0], [1 1], [2 2], (2 5], (5 10], (10 20], (20 100] }
	9-class	{ [0 0], [1 1], [2 2], [3 3], (3 5], (5 7], (7 10], (10 20], (20 100] }
RemainBins	3-class	{ [0 1], (1 5], (5 100] }
NextBin	3-class	{ [0 2], (2 5], (5 100] }

In consequence, the trained DASENet generates a 200-dimension embedding vector $\mathbf{e}_{\text{seq}}$ for an activity stream, by taking a bin-sequence $S^T = [B^1, B^2, ..., B^T]$ as an input and processing it via the two-staged merged and sequence networks.

VI. TASK LEARNING USING DASENET

In this section, we present the learning process for a task using DASENet and describe several task scenarios.

A. TASK LEARNING PROCESS

The activity stream embedding of DASENet allows us to build a task model data-efficiently, which can deal with bug-related temporal prediction problems. For a task, an additional network combined with DASENet can be separately trained with its own dataset and class labels. For implementing such an additional task network, we generally use a deep MLP function, yet with no limitation on how it is implemented. For example, we use $\mathsf{MLP}_{\text{task}}(\cdot)$ on the trained DASENet such as

$$\mathbf{y}_{\text{task}} = [y_1, \ldots, y_K] = \mathsf{MLP}_{\text{task}}(\mathbf{e}_{\text{seq}}) \tag{24}$$

where K is the same as the number of the task classes (e.g., K-class classification). To calculate the predicted probability of jth element (y_j) in $\mathbf{y}_{\text{task}}$, we use the softmax function (in (22)). Subsequently, for top-k accuracy-based prediction, we choose a k-size set of those elements with the highest probabilities.

B. TASK SCENARIOS

Here, we consider three task scenarios that involve analyzing bug-related activities to make temporal event prediction: bug-fix time (FixTime), the number of remaining activity bins (RemainBins), and the occurrence time of next activity bin (NextBin). While the activity stream embedding is based on FixTime, K-class classification (with various K-settings) tests and cross-project tests are discussed here. The other RemainBins and NextBin tasks are also related to project managers' jobs, similar to FixTime.

Table 4 illustrates the multi-class specification used in Section VII. For simple explanation, suppose that we have a task function $\mathsf{W}_{\text{task}}(R)$ that yields a certain task-specific value for a bug report R. Then, we represent class conditions by intervals $[a, b]$ (or $(a, b]$) where all the bug reports R such that $a \leq \mathsf{W}_{\text{task}}(R) \leq b$ (or $a < \mathsf{W}_{\text{task}}(R) \leq b$) are in the same class. The implementation of $\mathsf{W}_{\text{task}}(\cdot)$ is fully dependent on specific task semantics. Note that the 2-class criterion of FixTime is set to the median of bug-fix times of each project dataset, but otherwise the same criterion applies to all datasets; for the 2-class, we hardly found such a common criterion that can render all the datasets well-balanced.

1) FixTime

In general, it is important for project managers to estimate when bugs or issues are resolved. In the context of bug-fix time prediction, a task function of a bug report R is represented as a daily-basis time gap from the latest update on R (i.e., day for the last bin) to when R is closed. For generality, we assume K-class classification problems for various K settings, as illustrated in Table 4.

2) RemainBins

The RemainBins task is to predict how many activity bins will be generated more until a bug is closed. In the same way as FixTime, we use per-day activity bins and accordingly day-granularity.

3) NextBin

To predict when next bug-related activities will occur, the NextBin task uses a model with the same day-granularity above.

VII. EVALUATION

In this section, we evaluate the performance of our proposed DASENet model. For evaluation, we implemented the models using the Keras machine learning framework [42], and conducted tests on a system with an Intel CPU i9-9940X processor 3.30 GHz (14 cores), 128G of RAM, and NVIDIA

Geforce GTX 2080 SLI. The performance metric for evaluating a model with a G-sized dataset is Top-k accuracy $= \frac{1}{G}\sum_{g=1}^{G} \mathsf{match}\left(c_g, P_g\right)$ where c_g and P_g denote a ground truth class label and a set of predicted class labels respectively for each sample. Here, $\mathsf{match}\left(c_g, P_g\right) = 1$ if $c_g \in P_g$; 0, otherwise.

All the tests were performed with 10-fold cross-validation sets [43] and the average performance was summarized.

A. DATASETS

We utilize datasets collected from the bug tracking systems of open source software development projects including Firefox [26], Chromium [27], and Eclipse [28], which have been frequently used for bug-mining research. We use only bug report samples of which state is resolved and their lifetime is in the range from 1 to 100 days. We observed that some bugs were rapidly closed in a few seconds just after they were reported to the system, and others had remained with no activity for more than 100 days since there were open. These outlier data patterns on bug reports were also mentioned in the previous research [22]. Some might be interested in seizing characteristics of such abnormality, but we rather focus on modeling with normal data, similarly to [22], [44]. Our refined datasets that contain bug-related activities in the daily-basis sequence format can be accessed from Github [29].

Table 5 presents the statistics of the datasets. For example, in the Chromium dataset, there are 15,170 bug reports with 70,001 sequence samples which are observed from March 2014 to August 2015. In addition, the sample ratio of each dataset for training and testing is set to 6.4:3.6. Table 6 illustrates the statistics of the datasets with respect to the three tasks previously explained in VI-B.

TABLE 5. Datasets of studied open source projects.

Project	*Observation Period*	*Report #*	*Samples # (Train Samples #)*
Eclipse	Jan. 2010 - Mar. 2016	16,575	70,004 (44,545)
Chromium	Mar. 2014 - Aug. 2015	15,170	70,001 (44,801)
Firefox	Apr. 2014 - May 2016	13,619	70,001 (44,800)

TABLE 6. Dataset statistics with respect to three tasks.

Task	*Project*	*Median*	*Average*	*Standard Deviation*
FixTime	Firefox	3	7.8	12.6
	Chromium	4	11.1	16.8
	Eclipse	5	13.1	19.2
RemainBins	Firefox	2	2.5	2.1
	Chromium	2	2.2	1.9
	Eclipse	2	1.9	1.8
NextBin	Firefox	1	3.4	7.1
	Chromium	2	5.0	9.5
	Eclipse	2	7.0	12.3

B. MODELS IN COMPARISON

For comparison, we implemented several deep learning-based models including the baseline (DeepBase) and state-of-the-art models, i.e., DeepTriage [4], DeepLoc [11], in addition to our proposed DASENet. There has been no deep learning-based work introduced for bug-fix time prediction yet, so we leverage those two state-of-the-art models recently introduced in the domain of bug triage and bug localization.

Fig. 7 illustrates the structure and input log type of models in comparison. As shown, in terms of the input log type, while DeepBase and DeepTriage directly take textual data in bug reports, DeepLoc uses additional metadata. In terms of the network structure, DeepBase, DeepTriage, and DeepLoc exploit MLP, BLSTM, and CNN respectively for feature extraction. DASENet processes data streams of three log types from bug reports, and feeds each to its respective network, hence extracting relevant features separately first and combining them later.

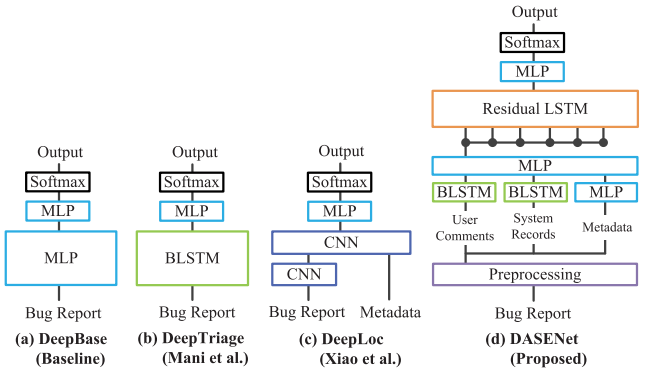


FIGURE 7. Comparison of learning model structures.

C. MODEL PERFORMANCE - ACCURACY

Table 7 provides an overall performance comparison based on top-1 and top-2 accuracy for K-class FixTime classification. In the table, the highest accuracy among all of the models is highlighted in bold, and the average accuracy difference between the DASENet and the other models is written in parentheses. Note that the class definition is described in Table 4.

Overall, DASENet outperforms the others in top-1 and top-2 accuracy. For example, the average performance gains of DASENet over the other models are 4.6 to 8.5 % in top-1 accuracy and 5.9 to 10.3 % in top-2 accuracy for the Firefox dataset. For all the datasets and K-classes (where $K = 2, 3, 5, 7, 9$), the gains of DASENet are 3.1 to 8.5 % in top-1 and 2.9 to 10.3 % in top-2 accuracy. This result indicates DASENet's capability using the two-staged learning process for continuously extracting and analyzing features over log streams via different networks.

Fig. 8 illustrates the top-1 accuracy of 3-class FixTime classification with respect to the bin-sequence length. As a bin-sequence gets longer, the top-1 accuracy of DeepBase and DeepTriage often tend to be lower; however, DeepLoc and

TABLE 7. FixTime — Top-1 and Top-2 accuracy of K-class classification.

Project	*Model*	*2-class*	*3-class*		*5-class*		*7-class*		*9-class*		*Average (Gap)*	
		Top-1	*Top-1*	*Top-2*	*Top-1*	*Top-2*	*Top-1*	*Top-2*	*Top-1*	*Top-2*	*Top-1*	*Top-2*
Firefox	DeepBase	71.2	51.1	83.4	33.3	61.6	24.2	45.0	19.7	37.2	39.9 (-4.6)	56.8 (-6.1)
	DeepTriage	66.8	43.6	78.4	30.7	57.5	20.7	39.4	18.6	34.9	36.1 (-8.5)	52.6 (-10.3)
	DeepLoc	71.2	50.5	83.4	34.4	61.6	23.2	43.5	20.1	39.5	39.9 (-4.7)	57.0 (-5.9)
	DASENet	**73.7**	**57.5**	**89.2**	**39.0**	**68.9**	**27.6**	**50.1**	**24.9**	**43.4**	**44.5** (·)	**62.9** (·)
Chromium	DeepBase	68.7	48.9	80.2	31.3	56.0	23.8	41.7	21.1	37.8	38.8 (-3.1)	53.9 (-4.4)
	DeepTriage	62.7	42.6	75.0	27.2	51.7	21.9	38.1	18.2	32.9	34.5 (-7.4)	49.4 (-8.9)
	DeepLoc	66.8	47.1	79.7	28.0	55.1	24.7	42.0	21.0	37.9	37.5 (-4.4)	53.7 (-4.6)
	DASENet	**69.8**	**54.4**	**86.0**	**34.4**	**60.7**	**27.3**	**46.7**	**23.5**	**39.9**	**41.9** (·)	**58.3** (·)
Eclipse	DeepBase	62.9	48.9	79.9	30.4	56.3	22.9	41.2	21.0	39.0	37.5 (-3.3)	54.1 (-2.9)
	DeepTriage	61.2	44.1	73.2	26.6	51.1	20.1	36.9	18.5	34.3	34.1 (-6.7)	48.9 (-8.1)
	DeepLoc	63.4	50.1	79.9	29.4	56.4	23.0	40.6	20.8	38.2	37.3 (-3.5)	53.8 (-3.2)
	DASENet	**65.5**	**53.4**	**83.1**	**34.4**	**59.8**	**26.2**	**44.6**	**24.5**	**40.6**	**40.8** (·)	**57.0** (·)

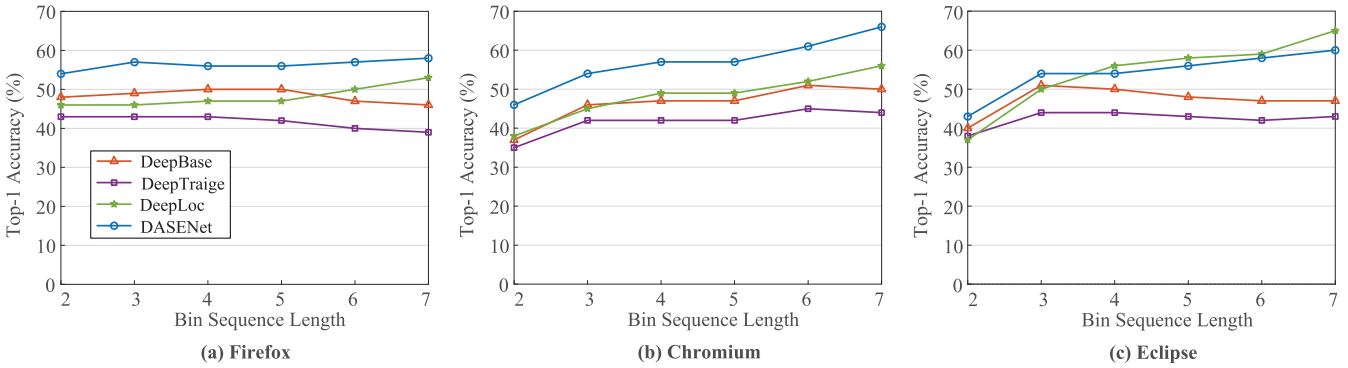


FIGURE 8. FixTime — Top-1 accuracy of 3-class classification w.r.t. bin-sequence length.

DASENet render different patterns, showing more robustness for bin-sequences of longer than 4. A bug report with a longer bin-sequence (i.e., having more activities overtime after a bug is reported) is likely to have a more complex history regarding how bug fixing jobs were made. Therefore, it might be effective to analyze temporal properties within a bin-sequence.

The semantic complexity of bug-related events was recently studied by Habayeb *et al*. [22], although their work relied on manual feature selection for HMM-based event sequence analysis. On the other hand, DASENet leverages advanced deep learning techniques, by taking the sequential form of time-series activity bins as input data for the RLSTM network. Hence, DASENet renders model accuracy more stable even for long-length bin-sequences. Interestingly, it is observed that DeepLoc yields a similar pattern with DASENet, but its accuracy is at least 5 % lower. Indeed, DeepLoc directly exploits the metadata including the bin-sequence length, and thus DeepLoc inherently confines itself to coarse-grain temporal analysis. In terms of analysis granularity, our DASENet contrasts to DeepLoc in that it is intended for fully exploring temporal properties of a bin-sequence.

D. MODEL PERFORMANCE - DATA EFFICIENCY

In this section, we discuss the stream embedding capability of DASENet by performing several cross-project tests and limited dataset tests. Here, we assume that there are a small number of labeled samples for a task, i.e., 2 to 10 % of an original training dataset used in the previous experiment (so, having 900 to 4,500 samples for model training).

1) CROSS-PROJECT TEST

Table 8 presents top-1 and top-2 accuracy of 3-class FixTime classification for "cross-project" cases, where a model is completely trained with a source project dataset

TABLE 8. FixTime — Cross-project tests.

Model	*Target: Eclipse*			
	Source: Firefox		*Source: Chromium*	
	Top-1	*Top-2*	*Top-1*	*Top-2*
DeepBase	41.7 (-8.6)	71.9 (-8.0)	38.1 (-12.2)	70.8 (-9.1)
DeepTriage	41.7 (-2.4)	72.1 (-1.1)	37.5 (-6.6)	69.3 (-3.9)
DeepLoc	45.7 (-4.4)	74.8 (-5.1)	44.7 (-5.4)	74.9 (-5.0)
DASENet	**55.2 (-1.2)**	**82.5 (-0.6)**	**52.7 (-0.7)**	**81.6 (-1.4)**

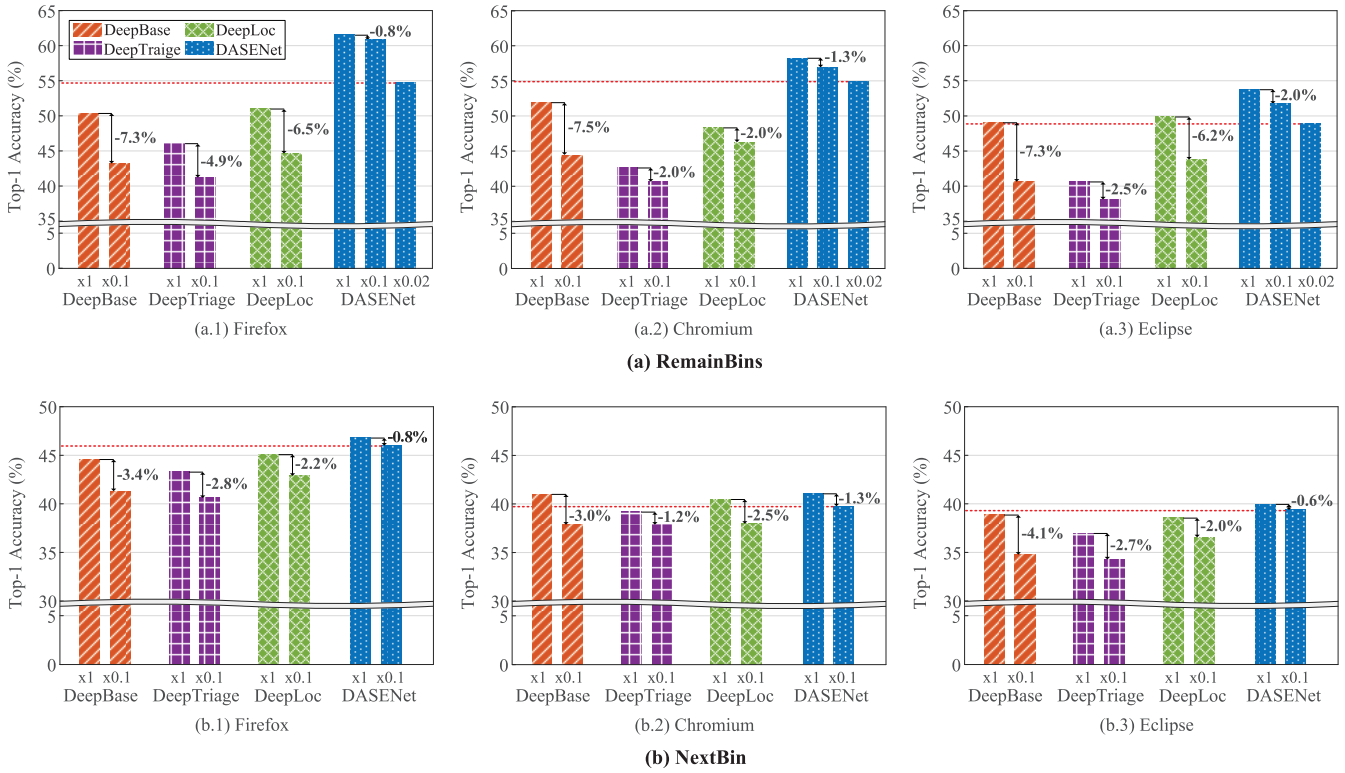


FIGURE 9. Variant tasks — 3-class classification performance with various sample amounts.

(e.g., 44,800 samples of Firefox) and the model is then fine-tuned with a small number of target project samples (e.g., 4,480 samples of Eclipse). In this test, we applied such a cross-project training process to all other models except for DASENet. For the DASENet model, we leverage its activity stream embedding in a way that we only train a target network on top of the pre-trained DASENet embedding network by using a small number of target project samples without end-to-end fine-tuning. In the table, we represent the accuracy drop from the same project test (previously explained in Table 7) to the cross-project test in parentheses. As shown, the DASENet model outperforms the other models, clearly showing its knowledge-transferable capability. Its small accuracy gap (-1.4 to -0.6 %) also represents that the knowledge established from the source dataset has a greater impact on the task than that of the other models.

Overall, the result of this cross-project test indicates that DASENet can be commonly used for a new software development project which might not have sufficient training samples.

2) VARIANT PREDICTION TASKS

Here, we discuss the general expressiveness of activity stream embedding by presenting how other prediction tasks about bug-related temporal events can be addressed. Similar to the cross-project tests above, we leverage the embedding output of DASENet based on the supervised learning for FixTime. However, we consider two different tasks, predicting the number of the remaining activities (RemainBins) and the occurrence time of the next activity (NextBin). We use the day-granularity of activity bins, same to FixTime.

Fig. 9 represents the top-1 accuracy of 3-class classification with various sample amounts. On the x-axis, "x#" represents the ratio to the original dataset size (i.e., 45,000 samples). For example, "x0.1" and "x0.02" mean that about 4,500 and 900 samples (10 % and 2 % of an original training dataset) were used for model training.

The DASENet model outperforms the others for all the cases, when the same number of samples were used (i.e., x1 and x0.1 cases in Fig. 9(a) and (b)). Furthermore, the DASENet model achieves stable performance, showing very marginal accuracy degradation no more than -2.0 % for x0.1 cases (e.g., as marked in the graph, -0.8 % for x0.1 for RemainBins of Firefox). This result is robust, in a contrast to the other models that show unstable performance with a variable accuracy degradation up to -7.5 % between x1 and x0.1 cases.

More importantly, the DASENet model using only about 900 training samples (x0.02) provides similar or better top-1 accuracy over the other models with about 45,000 training samples (x1) for RemainBins. Note that DASENet with x0.1 yields similar accuracy to the other models with x1 in the case of NextBin. This result demonstrates that the deep learning-based activity stream embedding can be effective for combining all activity logs to extract features, and thus the embedded output can be reused for data-efficiently training

a new task model related to bug-related activities. We also notice that the reusability or transferability level can be varying across different tasks and datasets, by observing that the top-1 accuracy gain of NextBin is less than that of RemainBins for the Chromium and Eclipse datasets. It was expected to some extent since the feature-embedded output of DASENet is likely to relate to the overall activity sequence of a bug report more than one specific future activity.

Fig. 10 depicts the time efficiency of training DASENet models for the RemainBins and NextBin tasks. In the figure, the learning speed gain is calculated by the training speed of DASENet over that of another model. As shown, the gains are greater than 1 for all the cases. This indicates the rapid learning convergence of DASENet, e.g., about 17 to 20 times (2.8 to 4.2 times) faster than DeepTriage (DeepLoc). DeepTriage exploits the BLSTM network with memory cells for extracting features from text data, which require intensive computation. It is worthwhile to note that the learning speed gain here mainly comes from the fact that the DASENet model renders its task network much simpler than DeepTriage, while it utilizes the pre-trained embedding model.

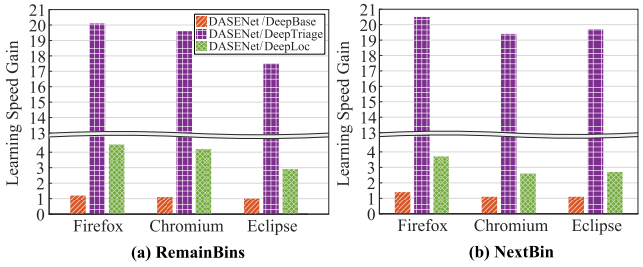


FIGURE 10. Variant tasks – Learning time efficiency.

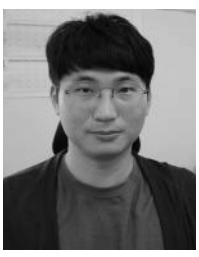
VIII. CONCLUSION

In this paper, we presented the deep learning-based model over log streams of bug-related activities for bug-fix time prediction. Our DASENet model achieved reliable prediction performance for the datasets of three open source projects. The model is two-staged with a merged network and a sequence network for handling different types of logging data over time. Moreover, the model of pre-trained activity stream embedding showed its advantages as a data-efficient learning process for cases where an insufficient number of samples were available for a new task. Our ongoing work is to investigate the model structure and extend it toward the joint learning with non-text activity logs, e.g., user-created diagram, in order to achieve a robust activity embedding strategy over heterogeneous data. Our implementation is currently confined to cover user-created comments, system-generated records, and statistical data, but it can be extended once additional networks have been properly designed for other log types. We are also working on applying our approach to a software development project that is internally operated by a platform company.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, Shanghai, China, May 2006, pp. 361–370.
[2] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Amsterdam, The Netherlands, Aug. 2009, pp. 111–120.
[3] 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 *Proc. 11th Joint Meeting Found. Softw. Eng.*, Paderborn, Germany, Sep. 2017, pp. 926–931.
[4] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. India Joint Int. Conf. Data Sci. Manage. Data*, 2019, pp. 171–179.
[5] J. A. Jones and M. J. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. 20th IEEE/ACM Int. Conf. Automated Softw. Eng.*, Long Beach, CA, USA, Nov. 2005, pp. 273–282.
[6] X. Sun, W. Zhou, B. Li, Z. Ni, and J. Lu, "Bug localization for version issues with defect patterns," *IEEE Access*, vol. 7, pp. 18811–18820, 2019.
[7] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Combining deep learning with information retrieval to localize buggy files for bug reports," in *Proc. 30th IEEE/ACM Int. Conf. Automat. Softw. Eng.*, Lincoln, NE, USA, Nov. 2015, pp. 476–481.
[8] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *Proc. 25th Int. Conf. Program Comprehension*, Buenos Aires, Argentina, May 2017, pp. 218–229.
[9] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin, "Improving bug localization with an enhanced convolutional neural network," in *Proc. 24th Asia–Pacific Softw. Eng. Conf.*, Nanjing, Chin, Dec. 2017, pp. 338–347.
[10] Y. Xiao, J. Keung, Q. Mi, and K. E. Bennin, "Bug localization with semantic and structural features using convolutional neural network and cascade forest," in *Proc. 22nd Int. Conf. Eval. Assessment Softw. Eng.*, Christchurch, New Zealand, Jun. 2018, pp. 101–111.
[11] Y. Xiao, J. Keung, K. E. Bennin, and Q. Mi, "Improving bug localization with word embedding and enhanced convolutional neural networks," *Inf. Softw. Technol.*, vol. 105, pp. 17–29, Jan. 2019.
[12] N. Jalbert and W. Weimer, "Automated duplicate detection for bug tracking systems," in *Proc. IEEE Int. Conf. Dependable Syst. Netw. FTCS DCC*, Anchorage, AK, USA, Jun. 2008, pp. 52–61.
[13] M.-J. Lin, C.-Z. Yang, C.-Y. Lee, and C.-C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *J. Syst. Softw.*, vol. 121, pp. 223–233, Nov. 2016.
[14] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "DWEN: Deep word embedding network for duplicate bug report detection in software repositories," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Companion*, Gothenburg, Sweden, May 2018, pp. 193–194.
[15] L. Yu, W.-T. Tsai, W. Zhao, and F. Wu, "Predicting defect priority based on neural networks," in *Proc. 6th Int. Conf. Adv. Data Mining Appl.*, Chongqing, China, Nov. 2010, pp. 356–367.
[16] Q. Umer, H. Liu, and Y. Sultan, "Emotion based automated priority prediction for bug reports," *IEEE Access*, vol. 6, pp. 35743–35752, 2018.
[17] C. Weiss, R. Premraj, T. Zimmermann, and A. Zeller, "How long will it take to fix this bug?" in *Proc. Fourth Int. Workshop Mining Softw. Repositories*, Minneapolis, MN, USA, May 2007, pp. 1–8.
[18] H. Zhang, L. Gong, and S. Versteeg, "Predicting bug-fixing time: An empirical study of commercial software projects," in *Proc. Int. Conf. Softw. Eng.*, 2013, pp. 1042–1051.
[19] L. Marks, Y. Zou, and A. E. Hassan, "Studying the fix-time for bugs in large open source projects," in *Proc. 7th Int. Conf. Predictive Models Softw. Eng.*, Banff, AB, Canada, Sep. 2011, pp. 11:1–11:8.
[20] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "Characterizing and predicting which bugs get fixed: An empirical study of microsoft windows," in *Proc. 32nd ACM/IEEE Int. Conf. Softw. Eng.*, Cape Town, South Africa, May 2010, pp. 495–504.
[21] E. Giger, M. Pinzger, and H. Gall, "Predicting the fix time of bugs," in *Proc. 2nd Int. Workshop Rec. Syst. Softw. Eng.*, Cape Town, South Africa, May 2010, pp. 52–56.
[22] M. Habayeb, S. S. Murtaza, A. Miranskyy, and A. B. Bener, "On the use of hidden Markov model to predict the time to fix bugs," *IEEE Trans. Softw. Eng.*, vol. 44, no. 12, pp. 1224–1244, Dec. 2018.
[23] P. Bhattacharya and I. Neamtiu, "Bug-fix time prediction models: Can we do better?" in *Proc. 8th Working Conf. Mining Softw. Repositories*, Honolulu, HI, USA, Jun. 2011, pp. 207–210.
[24] F. Zhang, F. Khomh, Y. Zou, and A. E. Hassan, "An empirical study on factors impacting bug fixing time," in *Proc. 19th Working Conf. Reverse Eng.*, Kingston, ON, Canada, Oct. 2012, pp. 225–234.

[25] P. Anbalagan and M. Vouk, "On predicting the time taken to correct bug reports in open source projects," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Edmonton, AB, Canada, Sep. 2009, pp. 523–526.
[26] *Firefox Project Bug Tracking System*. Accessed: Nov. 20, 2019. [Online]. Available: https://bugzilla.mozilla.org/home
[27] *Chromium Project Bug Tracking System*. Accessed: Nov. 20, 2019. [Online]. Available: https://bugs.chromium.org/p/chromium/issues/list
[28] *Eclipse Project Bug Tracking System*. Accessed: Nov. 20, 2019. [Online]. Available: https://bugs.eclipse.org/bugs/
[29] *Datasets for Bug-Related Activity Logs (Firefox, Chromium, Eclipse) on the GitHub*. Accessed: Nov. 20, 2019. [Online]. Available: https://github.com/mkris0714/Bug-Related-Activity-Logs.git
[30] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. 34th Int. Conf. Softw. Eng.*, Zürich, Switzerland, Jun. 2012, pp. 14–24.
[31] J. Lu, Y. Wei, X. Sun, B. Li, W. Wen, and C. Zhou, "Interactive query reformulation for source-code search with word relations," *IEEE Access*, vol. 6, pp. 75660–75668, 2018.
[32] T. Mikolov, K. Chen, G. S. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. Int. Conf. Learn. Represent.*, Scottsdale, AZ, USA, Jun. 2013, pp. 1–12.
[33] A. Graves, N. Jaitly, and A. Mohamed, "Hybrid speech recognition with deep bidirectional lstm," in *Proc. IEEE Workshop Autom. Speech Recognit. Understand.*, Olomouc, Czech Republic, Dec. 2013, pp. 273–278.
[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, Nov. 1997.
[35] F. A. Gers, N. Schraudolph, and J. Schmidhuber, "Learning precise timing with LSTM recurrent networks," *J. Mach. Learn. Res.*, vol. 3, pp. 115–143, 2002.
[36] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, Las Vegas, NV, USA, Jun. 2016, pp. 770–778.
[37] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," in *Proc. Int. Conf. Learn. Represent.*, San Diego, CA, USA, May 2015.
[38] J. Kim, M. El-Khamy, and J. Lee, "Residual LSTM: Design of a deep recurrent architecture for distant speech recognition," *CoRR*, 2017, Accessed: Jan. 10, 2017. [Online]. Available: http://arxiv.org/abs/1701.03360
[39] P. Oncharoen and P. Vateekul, "Deep learning for stock market prediction using event embedding and technical indicators," in *Proc. 5th Int. Conf. Adv. Inform., Concept Theory Appl.*, Krabi, Thailand, Aug. 2018, pp. 19–24.
[40] Y. Wang and J. Tang, "Event2Vec: Learning event representations using spatial-temporal information for recommendation," in *Proc. 23rd Pacific–Asia Conf. Knowl. Discovery Data Mining*, Macau, China, Apr. 2019, pp. 314–326.
[41] A. Conneau, D. Kiela, H. Schwenk, L. Barrault, and A. Bordes, "Supervised learning of universal sentence representations from natural language inference data," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, Copenhagen, Denmark, Sep. 2017, pp. 670–680.
[42] F. C. Keras. *Keras*. Accessed: Nov. 20, 2019. [Online]. Available: https://github.com/fchollet/keras
[43] R. Kohavi, "A study of cross-validation and bootstrap for accuracy estimation and model selection," in *Proc. 14th Int. Joint Conf. Artif. Intell.*, Montreal, QC, Canada, Aug. 1995, pp. 1137–1143.
[44] A. Lamkanfi and S. Demeyer, "Filtering bug reports for fix-time analysis," in *Proc. 16th Eur. Conf. Softw. Maintenance Reeng.*, Szeged, Hungary, Mar. 2012, pp. 379–384.



YOUNGSEOK LEE received the B.S. degree in electric and electronic engineering from Sungkyunkwan University, Suwon, South Korea, in 2013, where he is currently pursuing the integrated Ph.D. degree with the Department of Electrical and Computer Engineering. His research interests include intelligent application, software engineering, network traffic analysis, and wireless networks. He was a recipient of the Global Ph.D. Fellowship of Korea National Research Foundation, from 2013 to 2017.

SUIN LEE received the B.S. degree in computer engineering from Sungkyunkwan University, Suwon, South Korea, in 2018, where she is currently pursuing the M.S. degree with the Department of Platform Software. Her research interest includes machine learning and its application.

CHAN-GUN LEE received the B.S. degree in computer engineering from Chung-Ang University, Seoul, South Korea, in 1996, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejon, in 1998, and the Ph.D. degree in computer science from The University of Texas at Austin, Austin, TX, USA, in 2005. From 2005 to 2007, he was a Senior Software Engineer with Intel, Hillsboro, OR, USA. Since 2007, he has been a Professor with the Department of Computer Science and Engineering, Chung-Ang University, Seoul. His research interests include software engineering and real-time systems. He was a recipient of the Korea Foundation of Advanced Studies (KFAS) Fellowship, from 1999 to 2005.

IKJUN YEOM received the B.S. degree in electronic engineering from Yonsei University, Seoul, South Korea, in 1995, and the M.S. and Ph.D. degrees in computer engineering from the Texas A&M University, in 1998 and 2001, respectively. He was with DACOM Company, Ltd., from 1995 to 1996, and Nortel Networks, in 2000. He was an Associate Professor with the Department of Computer Science, KAIST, from 2002 to 2008. He is currently an Associate Professor with the Computer Engineering Department, Sungkyunkwan University. His research interests are in intelligent application, AQM, congestion control, wireless networks, and the future Internet architecture.

HONGUK WOO received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he worked for Samsung Research of Samsung Electronics as a Principal Engineer and the Vice President. Since 2018, he has been an Assistant Professor with the Department of Software, Sungkyunkwan University, Suwon, South Korea. His research interests include intelligent application, analytic monitoring, and networked cyber-physical systems.