The Journal of Systems and Software 158 (2019) 110406

Contents lists available at ScienceDirect

The Journal of Systems and Software

journal homepage: www.elsevier.com/locate/jss

DeepLink: Recovering issue-commit links based on deep learning

Hang Ruan[a,b,c], Bihuan Chen[a,b,c,*], Xin Peng[a,b,c], Wenyun Zhao[a,b,c]

[a] School of Computer Science, Fudan University, China
[b] Shanghai Key Laboratory of Data Science, Fudan University, China
[c] Shanghai Institute of Intelligent Electronics & Systems, China

ARTICLE INFO

Article history:
Received 21 December 2018
Revised 10 August 2019
Accepted 2 September 2019
Available online 3 September 2019

Keywords:
Issue-commit links
Deep learning
Semantic understanding

ABSTRACT

The links between issues in an issue-tracking system and commits resolving the issues in a version control system are important for a variety of software engineering tasks (e.g., bug prediction, bug localization and feature location). However, only a small portion of such links are established by manually including issue identifiers in commit logs, leaving a large portion of them lost in the evolution history. To recover issue-commit links, heuristic-based and learning-based techniques leverage the metadata and text/code similarity in issues and commits; however, they fail to capture the embedded semantics in issues and commits and the hidden semantic correlations between issues and commits. As a result, this semantic gap inhibits the accuracy of link recovery.

To bridge this gap, we propose a semantically-enhanced link recovery approach, named DeepLink, which is built on top of deep learning techniques. Specifically, we develop a neural network architecture, using word embedding and recurrent neural network, to learn the semantic representation of natural language descriptions and code in issues and commits as well as the semantic correlation between issues and commits. In experiments, to quantify the prevalence of missing issue-commit links, we analyzed 1078 highly-starred GitHub Java projects (i.e., 583,795 closed issues) and found that only 42.2% of issues were linked to corresponding commits. To evaluate the effectiveness of DeepLink, we compared DeepLink with a state-of-the-art link recovery approach FRLink using ten GitHub Java projects and demonstrated that DeepLink can outperform FRLink in terms of *F*-measure.

1. Introduction

Issues and commits are common software artifacts, which are managed through different process flows; i.e., issues are usually stored in bug-tracking systems such as Bugzilla, while commits are often stored in version control systems such as Git. Both of them contain a large amount of information about software evolution, but from different perspectives: issues capture the usage, while commits reflect the development. The links between issues and commits are important and valuable because they link two activities and can be leveraged to learn from the past and predict the future in various software engineering tasks.

For example, in bug prediction, issue-commit links are used to prepare training data and build prediction models (Bachmann et al., 2010; Rahman et al., 2013; Kim et al., 2007). In commit analysis, issue-commit links are analyzed to facilitate the characterization of the underlying intention and potential impact of code changes (Hindle et al., 2008; Purushothaman and Perry, 2005). In bug localization, issue-commit links are employed to construct localization models (Nguyen et al., 2011; Saha et al., 2013). In bug assignment, issue-commit links are required to build recommendation models for assigning a bug report to a developer (Anvik et al., 2006). In feature location, issue-commit links are used to establish mappings between source code and features (Dit et al., 2013).

In practice, issue-commit links are often manually established by developers; i.e., developers are required to include issue identifiers in commit logs (Śliwerski et al., 2005), especially for large projects such as Apache projects (Apache Software Foundation, 0000). Unfortunately, developers might not always follow such guidelines because they may forget or use irregular formats to include issue identifiers (Bachmann and Bernstein, 2009; Wu et al., 2011; Romo et al., 2014). As a result, only a small portion of links are established via issue identifiers, but a large portion of links are lost in the evolution history. However, it is a challenging, error-prone and time-consuming task to manually recover issue-commit links, even for experienced developers (Bachmann et al., 2010).

* Corresponding author at: School of Computer Science, Fudan University, China.
E-mail addresses: 16212010020@fudan.edu.cn (H. Ruan), bhchen@fudan.edu.cn (B. Chen), pengxin@fudan.edu.cn (X. Peng), wyzhao@fudan.edu.cn (W. Zhao).

https://doi.org/10.1016/j.jss.2019.110406
0164-1212/

To automate the recovery of issue-commit links, a commonly-used traditional heuristic is to search issue identifiers in commit logs (Fischer et al., 2003b; 2003a; Čubranić and Murphy, 2003; Śliwerski et al., 2005; Schröter et al., 2006). However, such a heuristic is often not sufficient to recover all issue-commit links, resulting in a biased data set of issue-commit links (Nguyen et al., 2010; Bird et al., 2009; Bachmann et al., 2010). Such a bias leads to a partial representation of the full population. As a result, the validity and generality of the approaches built on top of the biased data set are often compromised (Bachmann et al., 2010; Bird et al., 2014). For example, models of bug prediction (Bachmann et al., 2010; Rahman et al., 2013; Kim et al., 2007) and bug localization (Nguyen et al., 2011; Saha et al., 2013) might have evaluation biases or performance degradation; and findings from commit analysis (Hindle et al., 2008; Purushothaman and Perry, 2005) could become less generalizable.

To mitigate such a bias in recovered issue-commit links, several advanced recovery techniques have been recently proposed. They can be categorized into heuristic-based and learning-based techniques. Heuristic-based recovery (e.g., ReLink (Wu et al., 2011), MLink (Nguyen et al., 2012) and Schermann et al. (2015)) develops heuristic rules based on metadata matching (e.g., issue-resolving time and commit time should be close) and textual/code similarity between issues and commits to make them linked. Learning-based recovery (e.g., RCLink (Le et al., 2015), Rath et al. (2018), FRLink (Sun et al., 2017b) and PULink (Sun et al., 2017a)) first extracts metadata (e.g., assignee and committer) and textual/code similarity (e.g., issue title and commit log) features from issues and commits and then constructs a classification model to predict whether a link exists between an issue and a commit. However, since issues and commits are respectively described from the users' (e.g., how a bug is triggered and what its symptom is) and developers' (e.g., how the bug is resolved) perspective, these existing techniques fail to understand the embedded semantics in issues and commits, and lack the capability of capturing the hidden semantic correlations between issues and commits. Hence, this semantic gap between issues and commits inhibits the accuracy of the existing issue-commit link recovery techniques.

To fill the semantic gap, we propose a semantically-enhanced link recovery approach, named DeepLink, which is built upon deep learning techniques. Specifically, DeepLink applies word embedding (Mikolov et al., 2013a) to represent natural language descriptions and code in issues and commits, respectively, as word vectors that capture distributional semantics in issues and commits. With these word vectors, DeepLink adopts a neural network architecture based on recurrent neural network (Mikolov et al., 2010) to learn both the semantic representation of issues and commits and the semantic correlation between issues and commits. Therefore, the natural language descriptions or code in an issue have similar vectors to those in the commits that resolve the issue. When a pair of issue and commit arrives, DeepLink returns the probability that the issue and commit are linked. To further enhance the accuracy of DeepLink, we propose several heuristics to preprocess issues and commits for reducing the noise and improving the effectiveness.

To quantify the prevalence of missing issue-commit links, we analyzed 1078 GitHub Java projects, i.e., 583,795 closed issues, with more than 500 stars and 50 closed issues, and found that 42.2% of issues were linked to corresponding commits. Such a low percentage of linked issues to commits motivates the necessity of automatic link recovery approaches, while providing sufficient data for learning-based link recovery approaches.

To evaluate the effectiveness of DeepLink, we compared our approach with a state-of-the-art link recovery approach FRLink (Sun et al., 2017b) using ten GitHub Java projects. The results showed that our approach outperformed FRLink in terms of *F*-measure by 10.6% on average. By closely looking into the training and testing data of each project, we found that for 58.7% of issue-commit links, the issue titles were directly copied and used as the corresponding commit logs. After removing such biased data, four of the ten projects had too few data to be feasible for learning, and DeepLink outperformed FRLink in terms of F-measure by 4.6% in the remaining six projects on average. Further, we evaluated the effectiveness of the heuristics to preprocess issues and commits, and the experimental results indicated that the heuristics helped to improve *F*-measure by 2.1% on average.

In summary, this work makes the following contributions.

- We proposed a neural network-based approach, DeepLink, to automatically recover issue-commit links.
- We conducted a large-scale empirical study, using 1078 GitHub Java projects, to quantify the prevalence of missing issue-commit links.
- We perform extensive experiments using ten GitHub Java projects to evaluate the effectiveness of DeepLink by comparing a state-of-the-art link recovery approach FRLink.

The rest of the paper is structured as follows. Section 2 introduces the preliminaries on deep learning and gives a motivating example. Section 3 elaborates the details of our approach, and Section 4 evaluates our approach. Section 5 discusses the most closely related work before Section 6 draws the conclusions.

2. Deep learning and motivating example

In this section, we first briefly introduce several deep learning techniques that our approach is built upon, i.e., word embedding, recurrent neural network and long short-term memory. Then, we present an example to motivate our approach.

2.1. Word embedding

Word embedding is a technique to learn continuous fixed-length vector representations of words from a text corpus so that the vectors of similar words are close to each other in the vector space (Mikolov et al., 2013b; 2013a; Pennington et al., 2014). The word vectors can capture syntactic and semantic similarities between words (Mikolov et al., 2013a). For example, the word *big* is similar to *bigger* in the same sense that *small* is similar to *smaller*, while *France* is to *Paris* as *Germany* is to *Berlin*. Such a vector representation of words can be considered one of the key breakthroughs of deep learning on resolving challenging natural language processing tasks (Erhan et al., 2010).

Word2Vec (Mikolov et al., 2013a) and GloVe (Pennington et al., 2014) are the most popular methods to efficiently learn a standalone word embedding from a corpus of text. Two models, Continuous Bag-of-Words (CBOW) model and Skip-Gram model, are developed in Word2Vec to utilize the context of a word for learning the embedding. The context of a word is its surrounding words. For example, in the corpus *parse java files*, the words *parse* and *files* are the context of the word *java*. As shown in Fig. 1, the difference between the two models is that, the CBOW model predicts the probability of the current word based on its context, while the Skip-Gram model predicts the probability of the context given the current word. The word context is local as it is defined through a sliding window. Then, GloVe is proposed to combine the local context-based learning in Word2Vec with the global text statistics leveraged in matrix factorization techniques (e.g., Latent Semantic Analysis (Turney, 2005)). Empirical studies have demonstrated that the Skip-Gram model with negative sampling (Mikolov et al., 2013b) outperforms GloVe with less computation time (Levy et al., 2015).

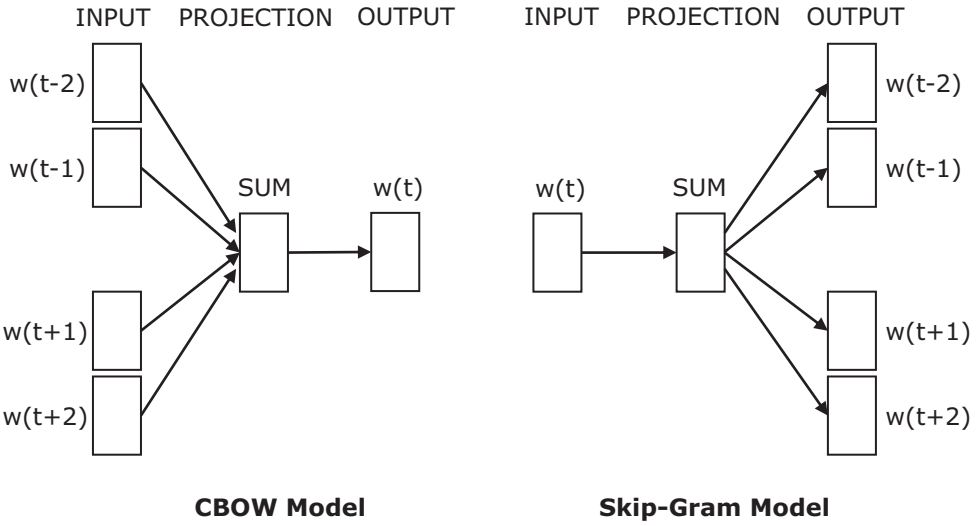


Fig. 1. CBOW model and skip-gram model from Mikolov et al. (2013a).

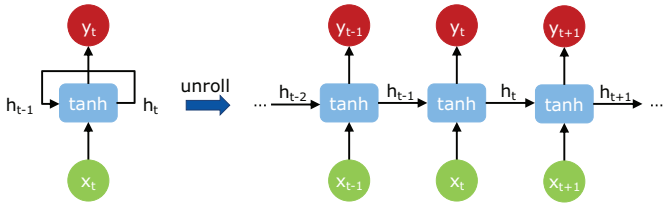


Fig. 2. Basic structure of RNN and its unrolled architecture.

2.2. Recurrent neural network

A recurrent neural network (RNN) (Mikolov et al., 2010) is a widely-used neural network for processing sequential data (e.g., text and audio). In RNNs, the input, hidden and output nodes are organized into layers, and the hidden node is recurrently used for computation. Therefore, the computation in the hidden node is based on both the input at the current time step and the hidden output from the previous time step. This creates an internal state (or memory) to capture temporal dynamic behaviors of sequential data.

Fig. 2 shows the basic structure of a RNN (in the left), which can be unrolled through time into a full network with a chained structure. The repeating module in the basic RNN structure contains an input node that maps each input into a vector, a hidden node that recurrently computes the hidden output vector (or the hidden state), and an output node that uses or outputs the hidden state. Formally, at time step t, given an input vector x_t and the previous hidden state h_{t-1}, a RNN computes the current hidden state h_t by Eq. (1),

$$h_t = tanh(W[h_{t-1}; x_t]) \tag{1}$$

where $W \in \mathbb{R}^{2d \times d}$ are the matrix of trainable parameters in the RNN, $[a; b] \in \mathbb{R}^{2d}$ represents the concatenation of two vectors, tanh is the activation function of the RNN, and d is the vector dimensions. The hidden states h_1, h_2, ... summarize the information about the sequential data; and a typical way is to select the last hidden state as the final output of the RNN.

2.3. Long short-term memory

One of the shortcomings of basic RNNs is that it is non-trivial for them to learn long-term dependencies in sequential data. For example, it is straightforward for a RNN to predict the last word in "fishes live in the *sea*" as the required context is close. However, it is difficult for a RNN to predict the last word in "I live in England... I speak *English*" because the required context of England for predicting English is very far away.

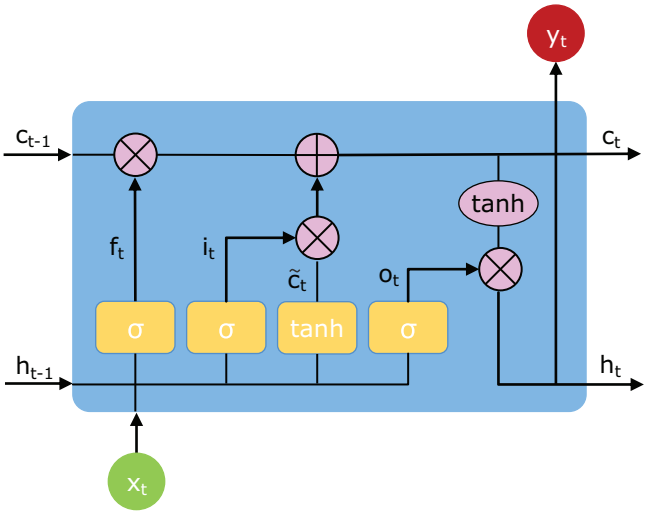


Fig. 3. Repeating module in LSTM.

To be capable of learning long-term dependencies, long short-term memory (LSTM) is introduced as a RNN variant (Hochreiter and Schmidhuber, 1997). In particular, the repeating module in a LSTM includes a cell state to record long-term dependencies, and utilizes three gates (i.e., a forget gate, an input gate and an output gate) to delete or add information to the cell state. A gate uses a sigmoid function σ and a point-wise multiplication operation $\times$ to control information throughput. The sigmoid function outputs a value between zero and one. Specifically, if the output is zero, a gate forbids any information from passing; and if the output is one, a gate allows all information to pass.

Fig. 3 illustrates the repeating module in a LSTM. The first step is to determine what information will be thrown away from the previous cell state c_{t-1}, which is realized through the forget gate by Eq. (2).

$$f_t = \sigma(W_f[h_{t-1}; x_t]) \tag{2}$$

Issue ID: #4483
Title: Pipeline counter is not truncated
Description: When the counter is too long, the counter is not truncated using ellipsis.

Instance:
a5a47d30fbdf2ad93f4cb12903d1401a290c1dab
Compare | Changes | VSM

Compare | Cha
Triggered by cha
on 05 Mar 2018

Creation Date: Mar 5, 2018
Close Date: Mar 6, 2018

(a) Example of an Issue in GoCD

Hash: 9f3e6269b1759497800e25b9538afdf66ebf41f2
Log: Truncate the label template on the dashboard (Fixes gocd#4483)
Commit Date: Mar 6, 2018
Changed Files: ...

(b) Example of a Commit in GoCD

Fig. 4. Example of an issue-commit link in GoCD.

The second step is to determine what information will be stored to the current cell state c_t. Hence, the input gate first determines what dimensions from inputs to use by Eq. (3), and then a *tanh* function computes a candidate cell state $\tilde{c}_t$ by Eq. (4).

$$i_t = \sigma(W_i[h_{t-1}; x_t]) \tag{3}$$

$$\tilde{c}_t = tanh(W_c[h_{t-1}; x_t]) \tag{4}$$

The third step is to update the current cell state c_t by Eq. (5); i.e., it first forgets the information it decides to forget in Eq. (2), and then adds the information it decides to use in Eqs. (3) and (4).

$$c_t = f_t \times c_{t-1} + i_t \times \tilde{c}_t \tag{5}$$

The final step is to compute the output based on the current cell state. It first decides what dimensions of the cell state to output via the output gate by Eq. (6), and then puts the current cell state through a *tanh* function and multiplies it by the output vector of the output gate by Eq. (7).

$$o_t = \sigma(W_o[h_{t-1}; x_t]) \tag{6}$$

$$h_t = o_t \times tanh(c_t) \tag{7}$$

Notice that $W_f, W_i, W_c, W_o \in \mathbb{R}^{2d \times d}$ are the matrix of trainable parameters, x_t is the input vector, h_{t-1} is the previous hidden state, $\times$ is point-wise vector multiplication, and + is point-wise vector addition.

2.4. A motivating example

Fig. 4 a and b show an issue[1] and a commit,[2] taken from the GoCD project on GitHub. GoCD is a continuous delivery server, which automates and streamlines the build-test-release pipeline for continuous delivery of a product. The issue included an issue ID, a short title, a long description, and the date of creating and closing the issue. The issue described a visualization problem of the *counter* of a pipeline instance; i.e., a long counter was failed to be truncated, whose symptom was reported by a snapshot in the description. The commit included a unique hash value, a log describing the purpose of the change, the commit date, and the changed files and their differences. As indicated by the issue ID in the commit log, the commit resolved the aforementioned issue by truncating the instance *label* in the template.

Interestingly, different terms (i.e., counter and label) were respectively used in the issue and commit to describe the same concept. As a result, a state-of-the-art learning-based technique FRLink (Sun et al., 2017b) fails to recover this issue-commit link because FRLink heavily relies on textual similarity and is unable to capture the semantic gap for counter and label. Differently, DeepLink utilizes deep learning techniques to bridge such a semantic gap, and successfully recover this link.

3. Methodology

Fig. 5 presents the approach overview of DeepLink, which consists of four steps: *data selection, data preprocessing, model training*, and *missing link recovering*. In particular, taking as inputs the issues and commits of a project (thus DeepLink is designed for in-project link recovery), our data selection step constructs the data set of true issue-commit links and false issue-commit links, which is used as the data set for deep learning. Then, our data preprocessing step extracts both text and code information from the issue and commit in each true and false link, applies multiple heuristics to reduce noise in the extracted information, and divides the data set into two sets: training data and testing data. Next, based on the training data, our model training step trains the neural network proposed for DeepLink. Finally, our missing link recovering step applies the trained model to the testing data to recover potential missing links. In the following subsections, we will elaborate each step in detail.

3.1. Data selection

A project can contain thousands of issues and commits, leading to a large space of potential links, especially for false links. To construct true links, existing link recovery approaches (e.g., Bird et al., 2010; Wu et al., 2011; Nguyen et al., 2012; Sun et al., 2017b) check whether the identifier of an issue s is included in the log of a commit m. If yes, a true link $\langle s, m \rangle$ is constructed. This method is commonly adopted for projects whose commits and issues are maintained in separate systems. GitHub projects, however, usually maintain issues and commits in the same eco-system, where developers are allowed to use one of the provided issue event, i.e., *referenced event*,[3] to reference an issue from a commit log. In other words, GitHub provides a mechanism of referenced events for developers to manually reference a related issue when submitting a commit. Hence, for GitHub projects, we develop a true link construction method by analyzing all the referenced events for each issue; i.e., for an issue s and each commit m that references the issue s, a true link $\langle s, m \rangle$ is constructed. Finally, we construct a set of true links L_T. Therefore, both searching issue identifiers in commit logs and leveraging referenced events can construct good true links that were actually manually established by developers during their programming activities.

To build false links, date information in issues and commits is usually used to reduce the space of potential links. For example, LINKSTER (Bird et al., 2010) constrains that the commit date is 7 days before or after the close date of an issue; MLink

[1] https://github.com/gocd/gocd/issues/4483.
[2] https://github.com/gocd/gocd/commit/9f3e626.
[3] https://developer.github.com/v3/issues/events/.

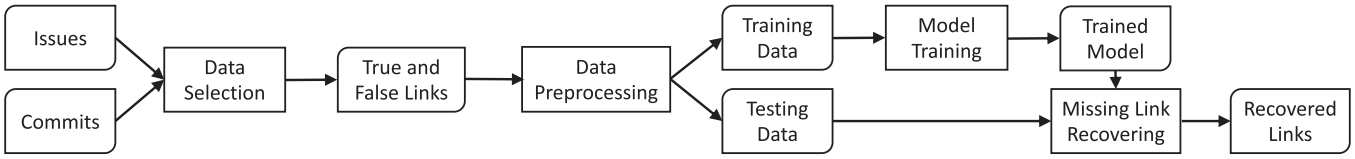


Fig. 5. Approach overview of DeepLink .

(Nguyen et al., 2012) requires that the commit date is between the creation date and close date of an issue; and FRLink (Sun et al., 2017b) requires that the commit date is 7 days before or after the creation, close, last modified, or comment date of an issue. As the false link construction is not the contribution of this paper, we directly follow the method in the previous work (Bird et al., 2010; Nguyen et al., 2012; Sun et al., 2017b). Specifically, for each commit m, we first locate each issue s such that the commit date of m is 7 days before or after the creation date or close date of s. We refer to such an issue set as S. Then, for each issue $s \in S$, we construct a potential link $\langle s, m\rangle$. If the link $\langle s, m\rangle$ is not in the true link set L_T but there exists a true link $\langle s', m\rangle$ $(s' \neq s)$ in L_T, the link $\langle s, m\rangle$ is established as a false link. The intuition is that, if a commit is already linked to some issue(s), there is a high chance that the commit is not truly linked to other issues. After this procedure, we establish a false link set L_F. As evidenced in previous works (Bird et al., 2010; Nguyen et al., 2012; Sun et al., 2017b), such a heuristic-based false link construction method provided a good data set. Hence, we directly follow previous works in constructing false links.

It is worth mentioning that, as the number of commits are often much larger than the number of issues, the number of established false links are often more than the number of constructed true links, as will be reported in our evaluation (Section 4.1). This results in an imbalanced data set of true and false links.

3.2. Data preprocessing

For a link $\langle s, m\rangle$ in L_T or L_F, the issue s and the commit m mainly include two types of information, i.e., text and code. In particular, the information in s contains an issue title, an issue description, comments, and the code in the description and comments; and the information in m contains a commit log and the code differences in changed code files. Considering the diversity and complexity nature of the information written by developers and users in issues and commits, we introduce several heuristics to preprocess the text and code from commits and issues so as to reduce the noise, instead of directly feeding such information to our neural network architecture. As issue comments can be very lengthy (e.g., clarifying reproduction environments or analyzing potential root causes) and may introduce noise, we currently do not use issue comments although they can also be informative.

3.2.1. Text preprocessing

The text information includes issue title, issue description and commit log. We apply three heuristics (i.e., removing hyperlinks, tags and code) and three commonly-used strategies (i.e., tokenizing, removing stop words and stemming) to preprocess the text. For those three heuristics, we give an example in the second column in Table 1, and apply a regular expression to realize the matching and processing, as shown in the last column in Table 1.

Removing Hyperlinks. Hyperlinks are frequently used in issue descriptions and commit logs for a variety of purposes (e.g., referencing a StackOverflow post to facilitate the issue discussion or a documentation page to clarity API usages). However, such hyperlinks, if not removed, are not informative, and will be split into several words in the later tokenizing step, introducing noise data to our text corpus for word embedding (see Section 3.3). Therefore, we remove hyperlinks.

Table 1
Heuristics of text preprocessing.

Type	Example	Regular Expr.
Hyperlinks	https://github.com/google/moe	http[s]?://[\s]+
Tags	[BEAM-5434]	\[[\s\S]*\]
Inline code	‘op.addOption’	‘[\s\S]*’
Large code	“‘if (isFull){return true;}”’	“‘[\s\S]*”’

Removing Tags. GitHub projects may specify a format to submit issues, especially for pull request issues; or issue reporters want to highlight the purpose of an issue. As a result, an issue title may start with a tag word in square brackets to link an issue in another issue tracking system or to indicate the issue purpose. For example, “[BEAM-5434]” in the issue title “[BEAM-5434] Improve error handling in the artifact staging service” is used to link the JIRA issue whose ID is 5434; and “[Discuss]” in the title “[Discuss] Consider using picocli as command line parser” is used to highlight the discussion purpose of this issue. However, such tags, if frequently used in issues, may become frequent words that are similar to stop words, i.e., they become semantically-related to too many words to be informative to find the context or the true meaning of a sentence. Hence, we remove such tags. On the other hand, only some projects use tags. Thus, we remove such tags for the purpose of simulating a general situation where this data is absent.

Removing Code. There often exists a lexical gap between code and natural language descriptions (McMillan et al., 2012; Ye et al., 2016). Thus, we distinguish text and code in our neural network architecture. To this end, we identify and remove embedded code in issue descriptions (which is used in Section 3.2.2). Code is styled in two main ways with GitHub's markdown. Specifically, ′ is used to mark inline code, and ′′′ is used to mark large code, as shown in the last two rows in Table 1.

Tokenizing, Removing Stop Words, and Stemming. We also apply the common strategies to preprocess issue titles, issue descriptions and commit logs. First, operators and punctuation are removed through tokenizing; and the words after tokenizing are combined to a text sequence. Then, normal English stop words such as *in* and *at* are removed from the text sequence by using the list of stop words in Porter (2011a). Last, the root form of each word in the text sequence is converted through stemming (Porter, 2011b).

3.2.2. Code preprocessing

We extract code information such as class names, method names, and variable names from code differences in commits and embedded code in issue descriptions (identified in Section 3.2.1). In particular, we use the patterns of code terms in Nguyen et al. (2012) and Sun et al. (2017b) to extract code terms. Table 2 reports all the patterns, which cover common naming conventions/styles in Java and C/C++ projects. After extracting code terms, we obtain a sequence of code terms for each commit and issue. Then, for the code terms that match the *CamelCase* pattern, we split them because they contain not only one word. For example,

Table 2
Code term extraction patterns (Taken from Sun et al. (2017b)).

Type	Example	Regulation Expr.
C_Notation	OPT_INFO	[A-Za-z]+[0–9]*_.*
Qualified name	op.addOption	[A-Za-z]+[0–9]*[\.].+
CamelCase	addToList	[A-Za-z]+.*[A-Z]+.*
UpperCase	XOR	[A-Z0-9]+
System variable	_cmd	_+[A-Za-z0-9]+.+
Reference expression	std::env	[A-Za-z]+[:]{2,}.+

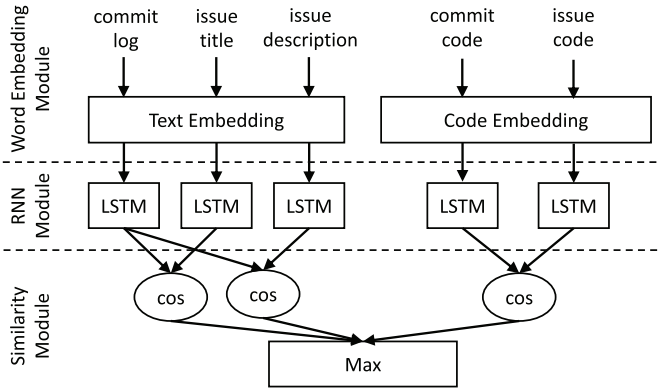


Fig. 6. The neural network architecture in DeepLink .

the code term *addToList* is split to three words: *add, to* and *list*. Finally, tokenizing, removing stop words and stemming are also applied to the code sequence.

Notice that the text sequences and code sequences after the previous text and code preprocessing may contain words whose length is less than two or words that only contain digits, which are not informative but may introduce noise. Therefore, we remove such words. Finally, for each link $\langle s, m\rangle$, $s = \langle E, P, Q\rangle$ and $m = \langle G, R\rangle$, where the issue title E, the issue description P and the commit log G are respectively represented as a text sequence; and the commit code Q and the issue code R are respectively represented as a code sequence.

3.3. Model training

Before introducing the details of model training, we present the neural network architecture used in DeepLink.

3.3.1. Neural network architecture

Fig. 6 presents the overall architecture of the deep neural network in DeepLink. The network consists of three modules: word embedding module, RNN module, and similarity module. Our word embedding module re-presents text information and code information as word vectors to capture distributional semantics in issues and commits from the perspective of text and code, respectively. Our RNN module learns the semantic representation of an issue and a commit. Our similarity module learns the semantic correlation between an issue and a commit. Next, we introduce each module in detail.

Word Embedding Module. As we distinguish text information and code information, we design two word embedding networks in our architecture: text embedding and code embedding. Hence, we need a text corpus and a code corpus. By traversing each link $\langle s, m\rangle$ in the true link set L_T and false link set L_F, we put the issue title E and issue description P of s and the commit log G of m into a text corpus, and put the issue code Q of s and the commit code R of m into a code corpus. Then, we use the Skip-Gram model, as introduced in Section 2.1, to train and build the text embedding network and code embedding network.

After word embedding, each word in issue title E, issue description P, issue code Q, commit log G and commit code R is denoted as a d-dimensional word vector; i.e., $E = e_1, \ldots, e_{E_N}$, $P = p_1, \ldots, p_{P_N}$, $G = g_1, \ldots, g_{G_N}$, $Q = q_1, \ldots, q_{Q_N}$ and $R = r_1, \ldots, r_{R_N}$ are represented as a sequence of word vectors.

RNN Module. Our RNN module respectively embeds the issue title E, issue description P, issue code Q, commit log G, and commit code R using five different RNNs. As introduced in Section 2.3, traditional RNNs cannot handle long-term dependencies, while LSTM can handle long-term dependencies. Hence, we use the special kind of RNNs, LSTM, in DeepLink. As the titles, descriptions and code of different issues and the logs and code of different commits have variable sequence lengths, we use zero padding on the inputs to LSTM.

Instead of introducing all five LSTMs, we only illustrate the LSTM for issue titles because they share the same structure but differ in the inputs, outputs, and trainable parameters. Consider a sequence of word vectors $E = e_1, \ldots, e_{E_N}$ of the issue title of an issue, the LSTM embeds the sequence of word vectors $e_1, \ldots, e_{E_N}$ by Eqs. (8)–(13), following the same procedure as introduced in Section 2.3.

$$f_t^E = \sigma(W_f^E[h_{t-1}^E; e_t]) \tag{8}$$

$$i_t^E = \sigma(W_i^E[h_{t-1}^E; e_t]) \tag{9}$$

$$\tilde{c}_t^E = tanh(W_c^E[h_{t-1}^E; e_t]) \tag{10}$$

$$c_t^E = f_t^E \times c_{t-1}^E + i_t^E \times \tilde{c}_t^E \tag{11}$$

$$o_t^E = \sigma(W_o^E[h_{t-1}^E; e_t]) \tag{12}$$

$$h_t^E = o_t^E \times tanh(c_t^E) \tag{13}$$

Here, $e_t \in \mathbb{R}^d$ $(t = 1, \ldots, E_N)$ is a word vector learned via our word embedding module, and $W_f^E, W_i^E, W_c^E, W_o^E \in \mathbb{R}^{2d \times d}$ are the matrix of trainable parameters.

Finally, the last hidden state $h_{E_N}^E$, $h_{P_N}^P$, $h_{Q_N}^Q$, $h_{G_N}^G$ and $h_{R_N}^R$ respectively represent the final vector of the issue title, issue description, issue code, commit log and commit code.

Similarity Module. As we want the vectors of text and code in issues are similar to the vectors of text and code in issue-fixing commits; i.e., for every true link, the vector of issue title $h_{E_N}^E$ and the vector of issue description $h_{P_N}^P$ should be similar to the vector of commit log $h_{G_N}^G$, or the vector of issue code $h_{Q_N}^Q$ should be similar to the vector of commit code $h_{R_N}^R$; and for every false link, they should be the other way around.

To this end, we use the cosine similarity to measure the similarity between an issue s and a commit m. The cosine similarity is defined in Eqs. (14) and (15).

$$cs(x, y) = \frac{x^T y}{\| x \| \| y \|} \tag{14}$$

$$cs_{\langle s,m\rangle} = max(cs(h_{G_N}^G, h_{E_N}^E), cs(h_{G_N}^G, h_{P_N}^P), cs(h_{R_N}^R, h_{Q_N}^Q)) \tag{15}$$

Three cosine similarities are calculated between commit log and issue title, between commit log and issue description, and between commit code and issue code. The maximum similarity among them is chosen as the final similarity $cs_{\langle s,m\rangle}$. The higher the similarity, the higher chance that there is a true link between an issue and a commit.

3.3.2. Neural network training

The goal of our model training is to make the issue s and the commit m in each true link from L_T have a high similarity, while making s and m in each false link from L_F have a low similarity.

Recall that the data set of true and false links is imbalanced; i.e., the number of false links L_F are much more than the number of true links L_T. To mitigate this imbalance, we use under-sampling (Liu et al., 2009) to create a balanced data set of false links $L'_F \subseteq L_F$; i.e., we randomly sample the same number of false links to true links. Then, L_T and L'_F are split into training and testing set according to the ratio of 4:1. The training set L_{TS} is used in our neural network training, and the testing set is used in our missing link recovery (see Section 3.4).

When trained on L_{TS}, our neural network architecture measures the cosine similarity of the issue s and commit m in each link $\langle s, m\rangle \in L_{TS}$ and minimizes the loss function in Eq. (16),

$$loss(\mathcal{W}) = \sum_{\langle s,m\rangle \in L_{TS}} | \, label_{\langle s,m\rangle} - cs_{\langle s,m\rangle} \, | \tag{16}$$

where $label_{\langle s,m\rangle}$ is 1 if $\langle s, m\rangle$ is a true link, and $label_{\langle s,m\rangle}$ is 0 if $\langle s, m\rangle$ is a false link; and $\mathcal{W}$ denotes the trainable parameters in our neural network architecture. Intuitively, the loss function makes the cosine similarity of a true link to be close to 1, while making the cosine similarity of a false link to be close to 0.

3.4. Missing link recovering

Given the trained model, when a link in the testing set arrives, our neural network architecture can return the similarity of the issue and commit in this link. If the similarity is larger than 0.5, we consider this link as a true link; otherwise, we consider it as a false link. Here 0.5 is established as a good threshold empirically. Similarly, when a new pair of issue and commit comes, we first applies pre-processing on the pair and then feed it to our trained model to determine whether the issue is linked to the commit or not.

4. Evaluation

We have implemented DeepLink in Python with 1.9K lines of code. In particular, we used the Word2Vec in gensim (Řehůřek and Sojka, 2010) to build the two word embedding networks, and TensorFlow (Abadi et al., 2015) to construct our RNN module and similarity module. The source code of DeepLink together with all the data set is available at our website.[4]

4.1. Evaluation setup

To quantify the prevalence of missing issue-commit links and evaluate the effectiveness of DeepLink, we designed experiments to answer the following three research questions.

- **RQ1:** How is the prevalence of missing issue-commit links in open-source projects? (Sections 4.2 and 4.5)
- **RQ2:** What is the effectiveness of DeepLink in recovering issue-commit links? (Sections 4.3 and 4.5)
- **RQ3:** What is the effectiveness of the data preprocessing step in DeepLink? (Section 4.4)

Subjects. To answer **RQ1**, we picked GitHub Java projects that had more than 500 stars and 50 closed issues. These two criteria on stars and issues are to ensure the quality of the projects and their frequent usage of issue tracking systems so that a representative set of projects was selected. Finally, we had a list of 1078 Java projects with 583,795 closed issues. On the other hand, to answer **RQ2** and **RQ3**, from the 1078 Java projects, we randomly selected ten projects whose stars were larger than 1000; and constructed the true links and false links as introduced in Section 3.1. Table 3 reports the statistics about the ten projects, including the number of stars, commits, issues, true links and false links. `Keycloak` is an identity and access management solution. `Checkstyle` is a checker of Java code for adherence to a code standard. `Pentaho Kettle` is an enterprise platform for accelerating data pipeline. `GoCD` is a continuous delivery server. `fabric8` is a microservices platform. `Closure Compiler` is a JavaScript checker and optimizer. `Flink` is a stream processing framework. `gRPC` is a remote procedure call (RPC) framework. `Beam` is a unified model for defining batch and streaming data-parallel processing pipelines. `CrateDB` is a distributed SQL database. The ten projects span across a diversity of application domains. We can see from the last two columns that, the number of false links is much larger than the number of true links, leading to an imbalanced data set. We use under-sampling to mitigate it (see Section 3.3.2).

Comparisons and Metrics. To answer **RQ2**, we chosen a state-of-the-art issue-commit link recovery approach FRLink (Sun et al., 2017b), and compared DeepLink with FRLink based on three metrics, i.e., precision, recall and *F*-measure. These three metrics have been widely used in evaluating the effectiveness of the existing link recovery approaches (Bachmann et al., 2010; Wu et al., 2011; Nguyen et al., 2012; Rath et al., 2018; Le et al., 2015; Sun et al., 2017b; 2017a; Rath et al., 2018). Notice that PULink (Sun et al., 2017a), to the best of our knowledge, is the only technique that has been empirically shown to outperform FRLink. However, PULink is not publicly available; and thus we failed to compare DeepLink with PULink. On the other hand, to answer **RQ3**, we disabled all the data preprocessing steps in DeepLink except for three common steps (i.e., tokenizing, removing stop words and stemming), and called this approach as DeepLink–. Then, we compared DeepLink– with DeepLink with respect to precision, recall and F-measure. Notice that, DeepLink is currently designed for in-project link recovery, which is the same to all previous work (Bachmann et al., 2010; Wu et al., 2011; Nguyen et al., 2012; Rath et al., 2018; Le et al., 2015; Sun et al., 2017b; 2017a; Rath et al., 2018). Thus, we con-

[4] https://github.com/ruanhang1993/DeepLink.

Table 3
Subject projects and date set of links.

Project	Stars	Commits	Issues	True links	False links
`Keycloak`	2204	10,932	5166	4157	268,198
`Checkstyle`	3620	7904	5753	2749	170,739
`Pentaho Kettle`	1593	20,210	5286	3328	215,276
`GoCD`	4089	7053	4673	4551	268,140
`fabric8`	1543	13,495	7042	4611	487,844
`Closure Compiler`	4127	12,763	1253	1197	19,973
`Flink`	3566	14,001	5893	3610	408,332
`gRPC`	4131	2880	3919	2458	151,392
`Beam`	1820	13,526	5223	5178	779,681
`CrateDB`	1984	8708	7196	4420	351,855

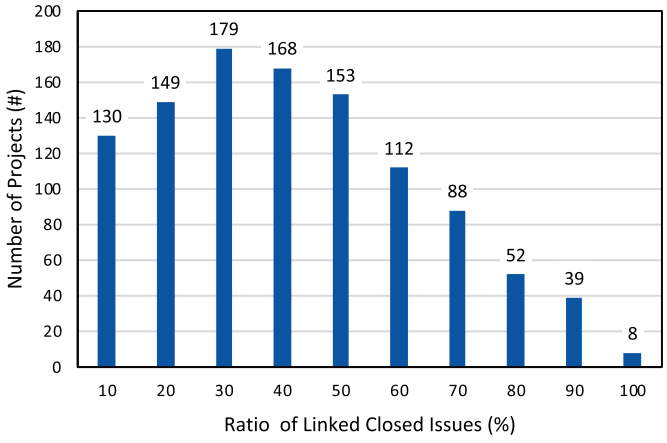


Fig. 7. Ratio of linked closed issues w.r.t. number of projects.

ducted in-project but not cross-project evaluation, i.e., we used L_T and L'_F constructed for each project as the data. Besides, as we split our data set into training and testing data according to the ratio of 4:1, we actually had five splits. Therefore, we conducted a five-folder cross validation, and reported the average results in our evaluation.

Evaluation Configuration. The configuration of our neural network architecture was set as follows. The number of dimensions of each word was set to 50 in our word embedding module; the hidden dimensions in LSTMs were set to 50; and the learning rate and batch size during our model training were respectively set to 0.001 and 128. All our experiments were conducted on a server machine with four 8-core 2.10 GHz Intel CPUs and 128 GB RAM. The GPU in the server machine is GeForce GTX 1080 with 12 GB memory.

4.2. Prevalence of missing issue-commit links (RQ1)

To quantify the prevalence of missing issue-commit links, we analyzed a total of 583,795 closed issues in 1078 open-source Java projects. Here we did not consider open issues because they often have not been linked to corresponding resolving commits. For each project, we measured the ratio of closed issues that were linked to resolving commits. Hereafter this ratio is referred to as *linked ratio* for simplicity. Notice that we leverage the referenced event in GitHub (see Section 3.1) to automatically decide whether a closed issue is linked to a commit. The results are shown in Fig. 7, where the *x*-axis denotes the linked ratio and the *y*-axis denotes the number of projects whose linked ratio isin a certain range; e.g., 179 projects had a linked ratio between 20% and 30%.

Numerically, for 458 (42.5%) projects, their linked ratio was below 30%; for 779 (72.3%) projects, their linked ratio was less than 50%; and only for 187 (17.3%) projects, their linked ratio was larger than 70%. Overall, around 42.2% of closed issues were linked to commits. These results indicate the poor practice of linking issues and commits in GitHub Java projects although GitHub provides a mechanism of referenced events to facilitate the linking, which motivates the necessity of automatically recovering the missed issue-commit links. On the other hand, the linked issues provide sufficient data to ease the feasibility of leveraging machine learning or deep learning techniques to recover issue-commit links.

> Missing issue-commit links are quite prevalence in GitHub Java projects; i.e., 42.5% of highly-starred projects had less than 30% of closed issues linked to commits, and 57.8% of closed issues were not linked to commits.

4.3. Effectiveness of DeepLink (RQ2)

To evaluate the effectiveness of DeepLink, we compared DeepLink with a state-of-the-art approach FRLink in terms of precision, recall and *F*-measure (here *F*1-measure is adopted to have a balance between precision and recall), using ten projects. Among all the 111,472 commits of these ten projects, 59,029 (52.9%) commits were linked to issues; and among the 59,029 issues, 39,302 (66.6%) commits referenced only one issue, and 19,727 (33.4%) commits referenced at least two issues. The results of the in-project link recovery are reported in the first seven columns of Table 4. The higher precision, recall or F-measure values are highlighted in bold. The *F*-measure improvement/degradation of DeepLink over FRLink is reported in parentheses in the seventh column.

In terms of precision, DeepLink outperformed FRLink in nine projects, and achieved an average improvement of 18.3%. With respect to recall, on the contrary, DeepLink outperformed FRLink in two projects, having an average degradation of 6.7%. On the whole, DeepLink achieved higher *F*-measure in eight projects and had an average F-measure of 86.1%, resulting in an average improvement of 10.6% over FRLink.

In detail, for the two projects where FRLink had a higher *F*-measure, DeepLink's degradation was slight. However, for the eight projects where DeepLink was better, DeepLink had a large improvement over FRLink in four of them. Specifically, our highest improvement was achieved for the projects `Closure Compiler` and `Beam`. It is worth mentioning that both DeepLink and FRLink achieved poor effectiveness on the project `Closure Compiler`. We believe that this is due to the small size of training data; i.e., only 1197 true links were constructed. Hence, for the projects that have a small number of true links, in-project learning may not be effective, and one potential solution is to leverage cross-project learning and recovery.

Table 4
Comparison of FRLink, DeepLink and DeepLink–.

Project	FRLink (%)			DeepLink (%)			DeepLink– (%)		
	Precision	Recall	*F*-Measure	Precision	Recall	*F*-Measure	Precision	Recall	*F*-Measure
`Keycloak`	93.9	93.7	93.8	**98.2**	**99.0**	**98.6** (+4.8)	96.9	98.6	97.7
`Checkstyle`	67.9	**90.1**	77.3	**80.9**	87.8	**84.2** (+6.9)	84.8	78.2	81.4
`Pentaho kettle`	91.1	**98.6**	94.7	**98.1**	95.2	**96.6** (+1.9)	95.7	97.1	96.4
`GoCD`	78.3	**89.6**	**83.6**	**86.5**	80.6	83.4 (−0.2)	91.2	78.0	84.1
`fabric8`	59.5	**88.6**	71.2	**76.8**	88.4	**82.2** (+11.0)	82.4	83.6	83.0
`Closure Compiler`	10.2	**87.5**	18.3	**62.5**	51.0	**55.5** (+37.3)	33.3	50.0	40.0
`Flink`	75.7	**97.5**	85.2	**85.9**	87.1	**86.5** (+1.3)	83.2	89.3	86.2
`gRPC`	71.3	86.5	78.0	**88.4**	**86.6**	**87.4** (+9.4)	87.4	87.2	87.3
`Beam`	43.3	**100.0**	60.4	**96.7**	90.5	**93.5** (+33.1)	93.5	91.6	92.5
`CrateDB`	**93.0**	**92.8**	**92.9**	92.9	92.3	92.6 (−0.3)	89.4	94.3	91.8
Average	68.4	**92.5**	75.5	**86.7**	85.8	**86.1** (+10.6)	83.8	84.8	84.0

Table 5
Results of biased data.

Project	Biased true links	Normal issues	PR issues	PR related true links
Keycloak	4035 (97.07%)	0	5166	4157
Checkstyle	1248 (45.40%)	2141	3612	1298
Pentaho Kettle	3122 (93.81%)	0	5286	3328
GoCD	2250 (49.44%)	2055	2618	2543
fabric8	1724 (37.39%)	2791	4251	2608
Closure Compiler	233 (19.47%)	962	291	252
Flink	1863 (51.61%)	0	5893	3610
gRPC	1568 (63.79%)	1417	2502	1864
Beam	1586 (30.63%)	0	5223	5178
CrateDB	3640 (82.35%)	601	6595	4369

In addition, we also measured the standard deviation of F-measure for the ten projects for DeepLink and FRLink. On average, DeepLink achieved a standard deviation of 1.8, while FRLink achieved a standard deviation of 2.5. This indicates that DeepLink is more stable than FRLink. Besides, we also measured the performance overhead of DeepLink and FRLink. On average, FRLink took about half an hour, while DeepLink spent around 3.2 h. Such a 5 × extra performance overhead of DeepLink is expected due to the nature of deep learning. Consider the effectiveness improvement, we believe DeepLink's performance overhead is acceptable.

DeepLink achieved higher precision but lower recall than a state-of-the-art approach FRLink. In general, DeepLink improved FRLink with respect to *F*-measure by 10.6% with acceptable performance overhead.

4.4. Effectiveness of preprocessing in DeepLink (RQ3)

To evaluate the effectiveness of our data preprocessing step in DeepLink, we compared DeepLink with DeepLink– where our preprocessing was disabled (as introduced in Section 4.1). The result is reported in the last three columns of Table 4.

We can see that DeepLink– achieved lower precision and F-measure than DeepLink on most of the projects. On average, after our preprocessing was disabled, the precision, recall and F-measure of DeepLink respectively decreased by 2.9%, 1.0% and 2.1%. This indicates that our data preprocessing on issues and commits is effective in improving DeepLink although the improvement is relatively slight.

Our data preprocessing in DeepLink helped to improved the F-measure of DeepLink by 2.1%.

4.5. In-depth analysis (RQ1 and RQ2)

According to the two recent works from Fu and Menzies (2017) and Liu et al. (2018), it is a good practice to (i) explore simple and fast approaches (Fu and Menzies, 2017) and (ii) investigate the reasons for good effectiveness (Liu et al., 2018) when leveraging deep learning techniques on software engineering tasks. As DeepLink is already baselined against the simple and fast approach FRLink (see Section 4.3), we further conducted an in-depth analysis of why DeepLink works well on issue-commit link recovery.

To this end, from each project, we randomly sampled 50 true links that DeepLink could correctly predict, and manually analyze the issue titles, issue descriptions, issue code, commit logs and commit code. Surprisingly, we found that in more than half of the sampled links, their issue titles were the same to the corresponding commit logs. It turns out that GitHub considers pull requests as a special type of issues, and thus our crawled issues included both normal issues and pull request issues. Commonly, a pull request is submitted together with commit(s), and users or developers might use the commit log as the issue title, which causes the bias in our data set of true links. This bias may lead to the artificial effectiveness of both DeepLink and FRLink. Hence, we measured the severity of the bias in the ten projects, and further analyzed its impact on **RQ1** and **RQ2**.

4.5.1. Bias severity

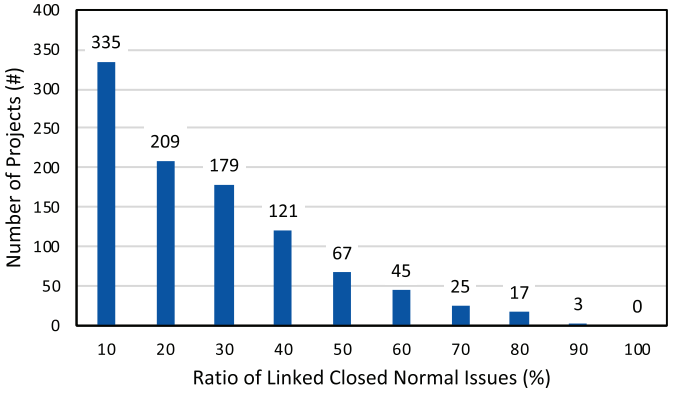
We first computed the ratio of biased true links where the issue title was the same to the commit log, the number of normal issues, and the number of pull request issues. The results are listed in the second to fourth columns of Table 5. Overall, 58.7% of true links had the same issue title and commit log. Specifically, the projects with higher ratio of pull request issues had more biased true links, e.g., Keycloak, Pentaho Kettle and CrateDB.

Then, we measured the number of true links whose issue is a pull request issue. The result is presented in the last column of Table 5. By subtracting the number of biased true links from the number of pull request-related true links, we found that around 27.1% of the pull request-related true links did not follow the practice of writing the same issue title and commit log.

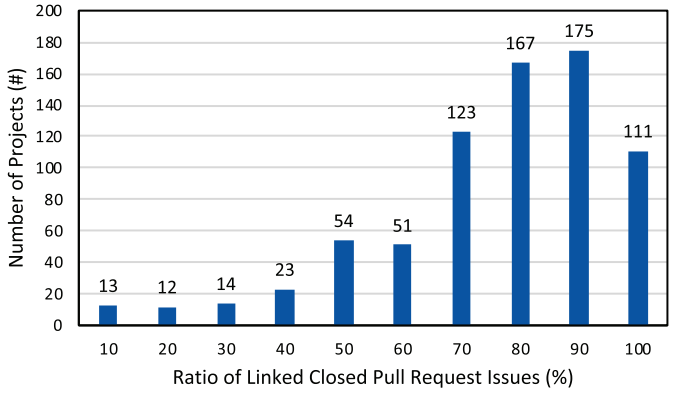
4.5.2. Bias impact

We first revisited **RQ1** by distinguishing between normal issues and pull request issues and measuring their respective linked ratio in the same way as in Section 4.2. The results are shown in Fig. 8a and b. Notice that 77 projects did not have normal issues, and 335 projects did not have pull request issues. As visually illustrated in Fig. 8, for 844 (84.3%) projects, their linked ratio for normal issues was below 40%; and for 576 (77.5%) projects, their linked ratio for pull request issues was above 70%. In total, there were 334,290 normal issues and 249,505 pull request issues; and 22.9% of normal issues were linked and 68.3% of pull request issues were linked. It is worth mentioning that the missing links for pull request issues are due to the deleted branches that contain the commit(s) submitted together with the pull requests.

Then, we revisited **RQ2** by removing the biased true links in Table 5 from the data set of true links in Table 3 and rerunning DeepLink and FRLink. However, the true links in Keycloak, Pentaho Kettle, gRPC and CrateDB became too few to be feasible for learning (i.e., 122, 206, 890 and 780). Therefore, only the other six projects were used. The results are reported in Table 6, where the higher precision, recall or *F*-measure values are highlighted in bold, and the effectiveness changes of FRLink and DeepLink from Table 4 are reported in parentheses. The findings from Table 4 are mostly hold in Table 6. On average, DeepLink outperformed FRLink by 4.6% in terms of *F*-measure. In detail, DeepLink outperformed FRLink in three of the six projects. For the three projects where FRLink had a higher *F*-measure, DeepLink's degradation was relatively small (i.e., 10.6% on average). However, for the three projects where DeepLink was better, DeepLink had a relatively large improvement over FRLink (i.e., 19.8% on average).



(a) Ratio of Linked Closed Normal Issues



(b) Ratio of Linked Closed Pull Request Issues

Fig. 8. Ratio of linked closed normal and pull request issues w.r.t. number of projects.

Table 6
Comparison of FRLink and DeepLink after removing biased true links.

Project	FRLink (%)			DeepLink (%)		
	Precision	Recall	*F*-Measure	Precision	Recall	*F*-Measure
`Checkstyle`	**73.7** (+5.8)	**86.8** (−3.3)	**79.7** (+2.4)	73.6 (−7.3)	70.3 (−17.5)	71.9 (−12.3)
`GoCD`	**43.1** (−35.1)	**94.7** (+5.0)	**59.3** (−24.3)	38.8 (−47.7)	73.2 (−7.4)	50.7 (−32.7)
`fabric8`	30.6 (−28.9)	**100.0** (+11.4)	46.9 (−24.3)	**57.3** (−19.5)	68.8 (−19.6)	**62.5** (−19.6)
`Closure Compiler`	16.7 (+6.4)	**64.9** (−22.6)	26.5 (+8.2)	**75.0** (+12.5)	50.0 (−1.0)	**60.0** (+4.5)
`Flink`	**90.5** (+14.8)	**88.8** (−8.7)	**89.6** (+4.4)	67.4 (−18.6)	82.2 (−4.9)	74.1 (−12.4)
`Beam`	32.1 (−11.2)	**100.0** (0.0)	48.6 (−11.8)	**75.2** (−21.5)	48.2 (−42.3)	**58.8** (−34.8)
Average	47.8 (−8.0)	**89.2** (−3.0)	58.4 (−7.6)	**64.6** (−17.0)	65.5 (−15.4)	**63.0** (−17.9)

Besides, after removing biased true links, we found that both DeepLink and FRLink suffered an effectiveness degradation, e.g., their F-measure respectively declined by 17.9% and 7.6%; and FRLink had a smaller degradation that DeepLink. One potential reason is that our deep learning-based approach is more sensitive to the size of training data. Overall, DeepLink had a *F*-measure of 63.0%, and there is still room for improvement. These results indicate the importance of digging deeply into the data to analyze why black-box learning-based approaches work for a fair evaluation. Inspired by Liu et al. (2018), we are the first, to the best of our knowledge, to conduct such an analysis for learning-based link recovery approaches.

Our data set of true links contained 58.7% biased true links where the issue title was the same to the commit log, which was caused by pull request issues. In 1078 projects, 22.9% of normal issues and 68.3% pull request issues were linked. After the biased true links were removed, DeepLink and FRLink respectively had a degradation of 17.9% and 7.6% in *F*-measure, while DeepLink still outperformed FRLink by 4.6%. However, there is still room for improvement.

4.6. Discussion

A main threat to the validity of our approach and evaluation is the quality and size of our data set. First, the true links are absolutely true links that were manually established by developers unless committers falsely referenced an issue when they committed because we used the referenced event provided by GitHub to construct true links. Second, during the false link construction, we adopted a threshold value of 7 days to select issues. However, it is actually a common good practice to adopt this threshold, as evidenced in previous works (Bird et al., 2010; Nguyen et al., 2012; Sun et al., 2017b). Hence, we directly follow previous works. Besides, the false links may be true links as we heuristically establish them. However, we believe the chance of falsely identifying a true link as false through our heuristics is low because our heuristics are similarly adopted and evidenced as good in the literature (e.g., Nguyen et al., 2012; Sun et al., 2017b). Third, our data set might still contain noise to hinder the effectiveness of learning, although we design several heuristics in our data preprocessing step to remove some noise. Further analysis for other potential noise in the data set is always needed to further improve our approach. Notice that we did not measure the impact of biased true links on the effectiveness of data our preprocessing (i.e., **RQ3**), because we believe our preprocessing is orthogonal to the bias problem. Fourth, the size of our data set of true links is not very large, and this problem will be even severe for small-scale projects, which limits the feasibility and applicability of our approach. Therefore, we are investigating the possibility of cross-project learning. The challenge is the diverse domain knowledge in different projects. Thus, domain-specific cross-project learning may be practical. Fifth, as GitHub does not provide any structured information to indicate how the resolution of an issue is in a general way, we currently only used closed issues in measuring the prevalence of missing links without considering their resolution types. As a result, the linked ratio might be higher than reported because there is no need for some closed issues to be linked to any commit as they might be duplicated or would not be fixed. While labels can be used to indicate such resolution types, only 3.25% projects use labels as reported by a recent large-scale study (Cabot et al., 2015). Thus, we also did not rely on labels to further filter issues.

The second threat to the validity of our evaluation comes from its scale. First, our prevalence evaluation of missing issue-commit links was conducted on 1078 highly-starred Java projects. We believe the prevalence is even severe in low quality projects; and further studies are needed to analyze the prevalence in projects written in other programming languages. Second, our effectiveness

evaluation was conducted on ten high-quality projects, whose scale is comparable to existing link recovery approaches; and it is interesting to investigate how our approach works on low-quality projects where incomplete or even in accurate commit logs might be possible. Besides, further studies on more high-quality projects are also needed to analyze the wide applicability of our approach.

The third threat to the validity is whether our learned model suffers the over-fitting problem. We followed the common practice of using the curves of loss value during training and validation (i.e., the loss curve in training and the loss curve in validation) to determine whether our learned model overfits. In detail, we observed all the curves in all the experiments, and found that both the loss curve in training and validation decreased and then stabilized, but the loss curve in validation had a small gap with the loss curve of training. As indicated by Anzanello and Fogliatto (2011), such a curve pattern reflects a good fit of the learned model. Therefore, we believe our model does not overfit.

The last threat to the validity is that we used under-sampling before cross validation. As a consequence, the under-sampled data might be not representative for the complete data. Hence, we under-sampled another different set of false links for each project and reran our approach. Our results indicated that the F-measure on these two datasets have only small differences, ranging from −1.2% to 1.5%. Therefore, we believe our evaluation is still representative.

5. Related work

We review the most closely related work in three aspects, i.e., recovering issue-commit links, recovering other links, and deep learning for software engineering.

Recovering Issue-Commit Links. To recover links between issues in a bug tracking system (e.g., Bugzilla) and commits in a version control system (e.g., Git), a traditional approach is based on the searching of issue identifiers in commit logs (Fischer et al., 2003b; 2003a; Čubranić and Murphy, 2003; Śliwerski et al., 2005; Schröter et al., 2006). However, several studies (Bird et al., 2009; Bachmann et al., 2010; Nguyen et al., 2010) have found that this traditional approach results in biased data of issue-commit links; i.e., it detects a small portion of links, but misses a large portion, because developers might not always include issue identifiers in commit logs. Thus, the validity and generality of the approaches (e.g., bug prediction (Bachmann et al., 2010; Kim et al., 2007) and bug localization (Nguyen et al., 2011; Saha et al., 2013)) built on top of such biased data are often compromised.

To address this problem, some heuristic-based approaches (Bachmann et al., 2010; Wu et al., 2011; Nguyen et al., 2012; Rath et al., 2018) have been recently proposed. Bachmann et al. (2010) and Bird et al. (2010) introduce LINKSTER to facilitate the manual identification of links. To fully automate the link recovery, Wu et al. (2011) develop ReLink through three heuristics: the issue-resolving time should be close to the commit time; the issue report should be similar to the commit log; and the developer who is responsible for an issue should be the committer. The thresholds for these heuristics are learned from true links. Bissyande et al. (2013) analyze ReLink and several information retrieval techniques, and pinpoint room for further improvements. Nguyen et al. (2012) propose MLink to further consider code heuristics: the code fragments or program entities in issue reports and commits (e.g., changed source code files) should be similar. Schermann et al. (2015) analyze the characteristics of Loners (i.e., one commit is linked to one issue) and Phantoms (i.e., several commits are linked to one issue) and use specific heuristics, similar to the ones in ReLink and MLink, to automatically establish these two kinds of links.

Moreover, several learning-based approaches (Le et al., 2015; Sun et al., 2017b; 2017a; Rath et al., 2018) have been proposed to leverage machine learning techniques to recover issue-commit links. Le et al. (2015) introduce RCLinker, which first applies Changescribe (Linares-Vásquez et al., 2015) to automatically generate commit logs because commit logs can be empty or short in size, extracts metadata and textual similarity features from those generated commit logs, commits and issues, and then constructs a classification model to predict whether a link exists between an issue and a commit. Similarly, Rath et al. (2018) use a different set of metadata and textual similarity features to create the model. Sun et al. (2017b, 2016) propose FRLink, which includes non-source documents (e.g., release notes) and excludes irrelevant source code files during feature extraction. Instead of relying on positive and negative links, Sun et al. (2017a) introduce PULink to use positive and unlabeled links. Evaluation results have shown that PULink can achieve competitive performance by utilizing only 70% positive links that are used in FRLink.

In summary, these heuristic-based and learning-based link recovery approaches lack the capability to capture the hidden semantic correlations between issues and commits, which leads to the semantic gap between issues and commits and thus inhibits the accuracy of link recovery. Instead, our approach is built on top of deep learning techniques to bridge the semantic gap and improve the link recovery accuracy.

Recovering Other Links. Apart from issue-commit links, links can be constructed among various software artifacts like requirements, source code and documentation (Cleland-Huang et al., 2014). For example, Antoniol et al. (2002) propose a probabilistic approach to construct links between source code and documentation, and Hayes et al. (2006) use Vector Space Model (Salton et al., 1975) with a thesaurus to build links between requirements and source code. A common challenge of these link recovery approaches arises from the semantic gap between different software artifacts. Thus, ontology (Jackson, 1998) is used to attempt to bridge the gap (Li and Cleland-Huang, 2013). However, it is time-consuming to build domain-specific ontologies. More recently, Guo et al. (2017) propose to use deep learning techniques to bridge the semantic gap between requirements and design. Our approach also relies on deep learning techniques, but targets a different link between issues and commits.

Deep Learning for Software Engineering. Deep learning techniques have been successfully applied to a variety of software engineering tasks. For example, Lam et al. (2015, 2017) combine deep learning with information retrieval techniques to identify buggy files in bug reports. Wang et al. (2016) use deep belief network to learn semantic features for bug prediction. White et al. (2016) rely on deep learning to link the patterns mined at the lexical level with the patterns mined at the syntactic level for the purpose of clone detection. Gu et al. (2016) adapt a RNN Encoder–Decoder model to generate API usage sequences for a natural language query. Pradel and Sen (2018) leverage deep learning to learn bug detectors for name-based bugs. Chen et al. (2018) apply deep reinforcement learning to test certificate verification in SSL/TLS implementations. Jiang et al. (2017) and Loyola et al. (2017) leverage deep neural networks to automatically generate commit messages. Gu et al. (2018) propose a deep neural network to search related code snippets for a natural language query. Our approach shares the deep learning nature with these approaches, but targets a different task, i.e., issue-commit link recovery.

6. Conclusions

In this paper, we proposed and implemented a semantically-enhanced approach, DeepLink, to recover issue-commit links. Specifically, we develop a neural network architecture, using word

embedding and recurrent neural network, to learn the semantic representation of issues and commits as well as the semantic correlation between issues and commits. Our evaluation on 1078 highly-starred GitHub Java projects (i.e., 583,795 closed issues) indicated that only 42.2% of issues were linked to corresponding commits. Our evaluation on ten GitHub Java projects demonstrated that DeepLink outperformed a state-of-the-art approach FRLink in terms of *F*-measure by 10.6% on average. After removing biased data, DeepLink still outperformed FRLink by 4.6%. In the future, we plan to conduct a further in-depth analysis of our data to remove noise and bias for further improving DeepLink.

Acknowledgment

This work was supported by the National Natural Science Foundation of China (Grant no. 61802067).

References

Abadi, M., Agarwal, A., Barham, P., Brevdo, E., Chen, Z., Citro, C., Corrado, G. S., Davis, A., Dean, J., Devin, M., Ghemawat, S., Goodfellow, I., Harp, A., Irving, G., Isard, M., Jia, Y., Jozefowicz, R., Kaiser, L., Kudlur, M., Levenberg, J., Mané, D., Monga, R., Moore, S., Murray, D., Olah, C., Schuster, M., Shlens, J., Steiner, B., Sutskever, I., Talwar, K., Tucker, P., Vanhoucke, V., Vasudevan, V., Viégas, F., Vinyals, O., Warden, P., Wattenberg, M., Wicke, M., Yu, Y., Zheng, X., 2015. TensorFlow: large-scale machine learning on heterogeneous systems. Software available from tensorflow.org.

Antoniol, G., Canfora, G., Casazza, G., De Lucia, A., Merlo, E., 2002. Recovering traceability links between code and documentation. IEEE Trans. Softw. Eng. 28 (10), 970–983.

Anvik, J., Hiew, L., Murphy, G.C., 2006. Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering, pp. 361–370.

Anzanello, M.J., Fogliatto, F.S., 2011. Learning curve models and applications: literature review and research directions. Int. J. Ind. Ergon. 41 (5), 573–583.

Apache Software Foundation,. How should i apply patches from a contributor? http://www.apache.org/dev/committers.html.

Bachmann, A., Bernstein, A., 2009. Software process data quality and characteristics: ahistorical view on open and closed source projects. In: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops, pp. 119–128.

Bachmann, A., Bird, C., Rahman, F., Devanbu, P., Bernstein, A., 2010. The missing links: bugs and bug-fix commits. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 97–106.

Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., Devanbu, P., 2009. Fair and balanced?: Bias in bug-fix datasets. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering, pp. 121–130.

Bird, C., Bachmann, A., Rahman, F., Bernstein, A., 2010. LINKSTER: enabling efficient manual inspection and annotation of mined data. In: Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 369–370.

Bissyande, T.F., Thung, F., Wang, S., Lo, D., Jiang, L., Reveillere, L., 2013. Empirical evaluation of bug linking. In: Proceedings of the 2013 17th European Conference on Software Maintenance and Reengineering, pp. 89–98.

Brindescu, C., Codoban, M., Shmarkatiuk, S., Dig, D., 2014. How do centralized and distributed version control systems impact software changes? In: Proceedings of the 36th International Conference on Software Engineering, pp. 322–333.

Cabot, J., Izquierdo, J.L.C., Cosentino, V., Rolandi, B., 2015. Exploring the use of labels to categorize issues in open-source software projects. In: Proceedings of the IEEE 22nd International Conference on Software Analysis, Evolution and Reengineering, pp. 550–554.

Chen, C., Diao, W., Zeng, Y., Guo, S., Hu, C., 2018. Drlgencert: deep learning-based automated testing of certificate verification in SSL/TLS implementations. In: Proceedings of the 34th International Conference on Software Maintenance and Evolution.

Cleland-Huang, J., Gotel, O.C.Z., Huffman Hayes, J., Mäder, P., Zisman, A., 2014. Software traceability: trends and future directions. In: Proceedings of the Future of Software Engineering, pp. 55–69.

Čubranić, D., Murphy, G.C., 2003. Hipikat: recommending pertinent software development artifacts. In: Proceedings of the 25th International Conference on Software Engineering, pp. 408–418.

Dit, B., Revelle, M., Gethers, M., Poshyvanyk, D., 2013. Feature location in source code: a taxonomy and survey. J. Softw. 25 (1), 53–95.

Erhan, D., Bengio, Y., Courville, A., Manzagol, P.-A., Vincent, P., Bengio, S., 2010. Why does unsupervised pre-training help deep learning? J. Mach. Learn. Res. 11, 625–660.

Fischer, M., Pinzger, M., Gall, H., 2003. Analyzing and relating bug report data for feature tracking. In: Proceedings of the 10th Working Conference on Reverse Engineering, pp. 1–10.

Fischer, M., Pinzger, M., Gall, H., 2003. Populating a release history database from version control and bug tracking systems. In: Proceedings of the International Conference on Software Maintenance, pp. 23–32.

Fu, W., Menzies, T., 2017. Easy over hard: a case study on deep learning. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, pp. 49–60.

Gu, X., Zhang, H., Kim, S., 2018. Deep code search. In: Proceedings of the 40th International Conference on Software Engineering, pp. 933–944.

Gu, X., Zhang, H., Zhang, D., Kim, S., 2016. Deep api learning. In: Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering, pp. 631–642.

Guo, J., Cheng, J., Cleland-Huang, J., 2017. Semantically enhanced software traceability using deep learning techniques. In: Proceedings of the 39th International Conference on Software Engineering, pp. 3–14.

Hayes, J.H., Dekhtyar, A., Sundaram, S.K., 2006. Advancing candidate link generation for requirements tracing: the study of methods. IEEE Trans. Softw. Eng. 32 (1), 4–19.

Hindle, A., German, D.M., Holt, R., 2008. What do large commits tell us?: A taxonomical study of large commits. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories, pp. 99–108.

Hochreiter, S., Schmidhuber, J., 1997. Long short-term memory. Neural Comput. 9 (8), 1735–1780.

Jackson, P., 1998. Introduction to Expert Systems. Addison-Wesley Longman Publishing Co., Inc..

Jiang, S., Armaly, A., McMillan, C., 2017. Automatically generating commit messages from diffs using neural machine translation. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, pp. 135–146.

Kim, S., Zimmermann, T., Whitehead Jr., E.J., Zeller, A., 2007. Predicting faults from cached history. In: Proceedings of the 29th International Conference on Software Engineering, pp. 489–498.

Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N., 2015. Combining deep learning with information retrieval to localize buggy files for bug reports (n). In: Proceedings of the 30th IEEE/ACM International Conference on Automated Software Engineering, pp. 476–481.

Lam, A.N., Nguyen, A.T., Nguyen, H.A., Nguyen, T.N., 2017. Bug localization with combination of deep learning and information retrieval. In: Proceedings of the 25th International Conference on Program Comprehension, pp. 218–229.

Le, T.B., Linares-Vasquez, M., Lo, D., Poshyvanyk, D., 2015. Rclinker: automated linking of issue reports and commits leveraging rich contextual information. In: Proceedings of the IEEE 23rd International Conference on Program Comprehension, pp. 36–47.

Levy, O., Goldberg, Y., Dagan, I., 2015. Improving distributional similarity with lessons learned from word embeddings. Trans. Assoc. Comput. Linguist. 3, 211–225.

Li, Y., Cleland-Huang, J., 2013. Ontology-based trace retrieval. In: Proceedings of the 7th International Workshop on Traceability in Emerging Forms of Software Engineering, pp. 30–36.

Linares-Vásquez, M., Cortés-Coy, L.F., Aponte, J., Poshyvanyk, D., 2015. Changescribe: a tool for automatically generating commit messages. In: Proceedings of the 37th International Conference on Software Engineering, pp. 709–712.

Liu, X.-Y., Wu, J., Zhou, Z.-H., 2009. Exploratory undersampling for class-imbalance learning. IEEE Trans. Syst. Man. Cybern. Part B 39 (2), 539–550.

Liu, Z., Xia, X., Hassan, A.E., Lo, D., Xing, Z., Wang, X., 2018. Neural-machine-translation-based commit message generation: how far are we? In: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, pp. 373–384.

Loyola, P., Marrese-Taylor, E., Matsuo, Y., 2017. A neural architecture for generating natural language descriptions from source code changes. In: Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers), pp. 287–292.

McMillan, C., Grechanik, M., Poshyvanyk, D., Fu, C., Xie, Q., 2012. Exemplar: a source code search engine for finding highly relevant applications. IEEE Trans. Softw. Eng. 38 (5), 1069–1087.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. CoRR. abs/1301.3781

Mikolov, T., Karafiát, M., Burget, L., Černockỳ, J., Khudanpur, S., 2010. Recurrent neural network based language model. In: Proceedings of the 11th Annual Conference of the International Speech Communication Association, pp. 1045–1048.

Mikolov, T., Sutskever, I., Chen, K., Corrado, G., Dean, J., 2013. Distributed representations of words and phrases and their compositionality. In: Proceedings of the 26th International Conference on Neural Information Processing Systems – Volume 2, pp. 3111–3119.

Nguyen, A.T., Nguyen, T.T., Al-Kofahi, J., Nguyen, H.V., Nguyen, T.N., 2011. A topic-based approach for narrowing the search space of buggy files from a bug report. In: Proceedings of the 2011 26th IEEE/ACM International Conference on Automated Software Engineering, pp. 263–272.

Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N., 2012. Multi-layered approach for recovering links between bug reports and fixes. In: Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering, pp. 63:1–63:11.

Nguyen, T.H.D., Adams, B., Hassan, A.E., 2010. A case study of bias in bug-fix datasets. In: Proceedings of the 2010 17th Working Conference on Reverse Engineering, pp. 259–268.

Pennington, J., Socher, R., Manning, C., 2014. Glove: global vectors for word rep-

resentation. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, pp. 1532–1543.
Porter, M. F., 2011a. An english stop word list.
Porter, M. F., 2011b. Snowball: a language for stemming algorithms.
Pradel, M., Sen, K., 2018. Deepbugs: a learning approach to name-based bug detection. In: Proceedings of the ACM on Programming Languages.
Purushothaman, R., Perry, D.E., 2005. Toward understanding the rhetoric of small source code changes. IEEE Trans. Softw. Eng. 31 (6), 511–526.
Rahman, F., Posnett, D., Herraiz, I., Devanbu, P., 2013. Sample size vs. bias in defect prediction. In: Proceedings of the 2013 9th Joint Meeting on Foundations of Software Engineering, pp. 147–157.
Rath, M., Rendall, J., Guo, J.L.C., Cleland-Huang, J., Mäder, P., 2018. Traceability in the wild: automatically augmenting incomplete trace links. In: Proceedings of the 40th International Conference on Software Engineering, pp. 834–845.
Řehůřek, R., Sojka, P., 2010. Software framework for topic modelling with large corpora. In: Proceedings of the LREC 2010 Workshop on New Challenges for NLP Frameworks, pp. 45–50.
Romo, B.A., Capiluppi, A., Hall, T., 2014. Filling the gaps of development logs and bug issue data. In: Proceedings of The International Symposium on Open Collaboration, pp. 8:1–8:4.
Saha, R.K., Lease, M., Khurshid, S., Perry, D.E., 2013. Improving bug localization using structured information retrieval. In: Proceedings of the 28th IEEE/ACM International Conference on Automated Software Engineering, pp. 345–355.
Salton, G., Wong, A., Yang, C.S., 1975. A vector space model for automatic indexing. Commun. ACM 18 (11), 613–620.
Schermann, G., Brandtner, M., Panichella, S., Leitner, P., Gall, H., 2015. Discovering loners and phantoms in commit and issue data. In: Proceedings of the 2015 IEEE 23rd International Conference on Program Comprehension, pp. 4–14.
Schröter, A., Zimmermann, T., Premraj, R., Zeller, A., 2006. If your bug database could talk.... In: Proceedings of the 5th International Symposium on Empirical Software Engineering, pp. 18–20.
Śliwerski, J., Zimmermann, T., Zeller, A., 2005. When do changes induce fixes? In: Proceedings of the 2005 International Workshop on Mining Software Repositories, pp. 1–5.
Sun, Y., Chen, C., Wang, Q., Boehm, B., 2017. Improving missing issue-commit link recovery using positive and unlabeled data. In: Proceedings of the 32Nd IEEE/ACM International Conference on Automated Software Engineering, pp. 147–152.
Sun, Y., Wang, Q., Li, M., 2016. Understanding the contribution of non-source documents in improving missing link recovery: an empirical study. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 39:1–39:10.
Sun, Y., Wang, Q., Yang, Y., 2017. Frlink: improving the recovery of missing issue–commit links by revisiting file relevance. Inf. Softw. Technol. 84 (C), 33–47.
Turney, P.D., 2005. Measuring semantic similarity by latent relational analysis. In: Proceedings of the 19th International Joint Conference on Artificial Intelligence, pp. 1136–1141.
Wang, S., Liu, T., Tan, L., 2016. Automatically learning semantic features for defect prediction. In: Proceedings of the 38th International Conference on Software Engineering, pp. 297–308.
White, M., Tufano, M., Vendome, C., Poshyvanyk, D., 2016. Deep learning code fragments for code clone detection. In: Proceedings of the 31st IEEE/ACM International Conference on Automated Software Engineering, pp. 87–98.
Wu, R., Zhang, H., Kim, S., Cheung, S.-C., 2011. Relink: recovering links between bugs and changes. In: Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering, pp. 15–25.
Ye, X., Shen, H., Ma, X., Bunescu, R., Liu, C., 2016. From word embeddings to document similarities for improved information retrieval in software engineering. In: Proceedings of the 38th International Conference on Software Engineering, pp. 404–415.

Hang Ruan received his BS.c. degree from Fudan University in 2016. He is a master student at the School of Computer Science in Fudan University, China. His research interests include software maintenance and evolution.

Bihuan Chen received his Ph.D. degree in Computer Science from Fudan University in 2014. He was a postdoctoral research fellow in Nanyang Technological University from 2014 to 2017. Since 2017, he has been a pre-tenure associate professor in Fudan University, China. His research interests lie in software engineering, focusing on program analysis, software testing, and software security.

Xin Peng received his Ph.D. degree in Computer Science from Fudan University in 2006. He is a full professor at the School of Computer Science in Fudan University, China. His current research interests include big code analysis, intelligent software development, software maintenance and evolution, and mobile computing.

Wenyun Zhao received his master's degree from Fudan University in 1989. He is a full professor of the School of Computer Science at Fudan University, China. His current research interests include software reuse, software product line, software component and architecture.