The Journal of Systems & Software 198 (2023) 111617

Contents lists available at ScienceDirect

The Journal of Systems & Software

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

Leveraging multi-level embeddings for knowledge-aware bug report reformulation[☆]

Cheng Zhou[a,b], Bin Li[a,*], Xiaobing Sun[a,*], Sheng Yu[a]

[a] School of Information Engineering, Yangzhou University, Yangzhou, China
[b] Taizhou University, Taizhou, China

ARTICLE INFO

Article history:
Received 22 July 2020
Received in revised form 20 October 2022
Accepted 15 January 2023
Available online 16 January 2023

Keywords:
Bug analysis
Bug report reformulation
Knowledge representation
Information retrieval

ABSTRACT

Software bug analysis based on the information retrieval (IR) technology is widely studied and used for bug understanding, localization and fixing. IR technology with various textual feature extraction methods formulates the textual information in a given new bug report (i.e., title and description) as an initial query. However, due to the low-quality content in the new bug report and improper representation to be used as a query, the retrieval results are usually not satisfactory.

To alleviate these problems, we propose a novel knowledge-aware bug report reformulation approach (a.k.a, KABR) by leveraging multi-level embeddings from the bug data. First, we construct a bug-specific knowledge graph (KG) to manage and reuse prior knowledge extracted from historical bug reports. Then, we extract word embedding from the original bug data, entity embedding and context embedding from the bug-specific KG to enhance the initial query. Finally, a new query representation is generated by leveraging multi-level embeddings through Convolutional Neural Networks (CNN) with the self-attention mechanism. We evaluate KABR based on the duplicate bug report detection task, and the experimental results show that KABR achieves 6%–11% F1-measure improvement over the state-of-the-art approaches.

1. Introduction

Software bugs are inevitable and seriously affect the quality of the software. Numerous software projects use software bug tracking systems (BTS) such as JIRA[1] and Bugzilla[2] to submit, track and manage bugs in the form of bug reports. A large number of bug reports are continuously submitted every day. For example, Mozilla's BTS[3] show that there were more than 250 new bugs submitted per day in 2020, and hundreds of new bugs reported in early 2020 have not been fixed. Table 1 shows two pair of bug reports with different quality information from the Mozilla project. Each bug report is given a unique ID, and mainly contains the title, description, comments, and the categorical bug information such as status, product and component, etc. Generally, when a developer is assigned a new bug, he/she needs to analyze the cause of the bug and develop a repair plan (Bacchelli and Bird, 2013) based on a short description of the bug symptoms. However, new bug reports often miss essential information and even contain incorrect information (Davies and Roper, 2014; Liu et al., 2020), moreover, the diversity and uneven quality of new bug reports make the bug analysis process tortuous (Kim and Jr., 2006). Therefore, for a large project consisting of hundreds or even thousands of files, understanding bugs, locating them, and fixing them become laborious and time-consuming.

Recently, there are many studies that help developers analyze new bugs by mining bug repositories to obtain relevant historical bug information, such as bug triage (Zhao et al., 2019a; Bodden et al., 2017), bug classification (Thung et al., 2012; Nayrolles and Hamou-Lhadj, 2018), bug localization (Zhang et al., 2019b; Xiao et al., 2019), bug prediction (Elmishali et al., 2019), etc. Researchers mainly apply the information retrieval (IR) technology with various textual feature extraction methods to formulate the full textual information in a given new bug report (i.e., title and description) as an initial query and automatically search for relevant bug files based on similarity matching (Ye et al., 2014; Ebrahimi et al., 2019; Yang et al., 2016; Hu et al., 2018; Ye et al., 2018; Xie et al., 2018; Zhao et al., 2019a). However, due to the low-quality content and limited information in the new bug report, the retrieval results are usually not satisfactory. Some approaches have been proposed to enrich the new bug report with additional information to enhance the retrieval effectiveness. These approaches fall into two categories: (1) *query expansion*, which mainly adds some co-occurring terms or synonyms

☆ Editor: Dr. Shane McIntosh.
* Corresponding authors.
E-mail addresses: lb@yzu.edu.cn (B. Li), xbsun@yzu.edu.cn (X. Sun).
1 https://www.atlassian.com/software/jira
2 https://www.bugzilla.org/
3 https://bugzilla.mozilla.org

https://doi.org/10.1016/j.jss.2023.111617
0164-1212/© 2023 Elsevier Inc. All rights reserved.

Table 1
Examples of bug reports from Mozilla's BTS.

Bug ID	108670[a]	108746[b]
Component	Composer	Composer
Product	SeaMonkey	SeaMonkey
Status	VERIFIED FIXED	VERIFIED DUPLICATE
Title	**Fix regressions caused** by **XUL syntax changes**.	**Crash opening List Properties dialog**.
Description (high-quality)	This is a critical bug since I have **data loss** because I can't **dismiss** this **dialog** to **save** and must **force quit**. When I open the **Page Title** and **Properties dialog**, the **window** comes up **blank** and covers the whole **screen**. I can't dismiss it. This is with yesterday's build on **Mac**. Kin also sees it with his build on **Windows**.	*Steps to reproduce*: **Open browser**, **Right-click**, select **Edit Page** in **Composer**. **Place cursor** in the text of the **list** of bugs. **Right-click** in the text, **select List Properties**. *Expected Results*: **Open** a **dialog** to **edit** the **list properties**. *Actual Results*: **Immediate crash**.
Bug ID	767056[c]	794399[d]
Component	SVG	SVG
Product	Core	Core
Status	VERIFIED FIXED	VERIFIED DUPLICATE
Title	**Crash** in **nsRegion::Or** (when I **test Bug 766956**, **browser crashes** when **resize browser**).	**SVG image loading causes firefox hangup** on i686 **linux.**
Description (low-quality)	*Steps to Reproduce:* Open http://granite.sru.edu/ddailey/svg/B/BBox0.svg. **Resize browser width** by **dragging window side border** quickly. Repeat step2 10–20 times. *Actual Results:* **Browser crashes** *Expected Results:* No **crash** Regression window(m-c) Not **crash**: http://hg.mozilla.org/mozilla-central/rev/f2b2b99108a2 (10 more) Suspected: **Bug 734082**, **Bug 614732**	*Steps to reproduce*: Please, open that url: http://www.blues.gda.pl/1.svg. *Expected Results*: **Firefox hangup** with **100% CPU usage**. Confirmed on i686 linux version. 64 bit version is not affected. *Actual Results*: **svg image displayed**.

[a] https://bugzilla.mozilla.org/show_bug.cgi?id=108670
[b] https://bugzilla.mozilla.org/show_bug.cgi?id=108746
[c] https://bugzilla.mozilla.org/show_bug.cgi?id=767056
[d] https://bugzilla.mozilla.org/show_bug.cgi?id=794399

from source code and external software engineering documents to the new bug report (Yang and Tan, 2014; Rahman and Roy, 2018b; Sisman and Kak, 2013; Rahman and Roy, 2017) ; (2) *feature enhancement*, which extracts multiple domain features as matching signals from related software engineering data or categorical bug information (Aggarwal et al., 2017; Ye et al., 2014; Hindle et al., 2016; Xie et al., 2018). The main shortcomings of these approaches are as follows:

- **Ignored semantic information during query expansion:** Existing query expansion approaches mainly utilize keywords in the initial query to obtain additional terms from pseudo-relevance feedback. But a lot of semantic information are not considered during this process, which generates unrelated or inaccurate query expansion results.
- **Improper query representation:** In the process of bug report reformulation, existing expansion approaches tend to treat the bug query as a sequence of terms, replace the keywords in the initial query with new terms from elsewhere or directly insert new terms into the initial query. These word-level fusion methods break the structural information of the initial query, which further affect the accuracy of retrieval results.

Information extraction (IE) technology, whose main task is to extract the factual knowledge from data of different structures and to realize effective use of data, has become a hot topic in many fields (Wang et al., 2017b; Li et al., 2018; Wang et al., 2019; Lin et al., 2017; Sabou et al., 2018). Based on IE, on the one hand, fine-grained knowledge information such as bug-specific entities and relations between entities could be extracted from the textual information in the bug report to comprehensively explore the latent semantics and associations within it. An entity is the smallest unit of knowledge. As highlighted in bold in Table 1, it can be a word or a phrase, with flexible forms. On the other hand, the entity-relation-entity triple could be used as the basic structure to further construct a bug-specific knowledge graph (KG), which provides a new way for the representation, management, and efficient reuse of massive bug data in the bug repository.

In this paper, we propose a novel knowledge-aware bug report reformulation approach (a.k.a, *KABR*) by leveraging multi-level embeddings to alleviate the above two problems. First, we construct a bug-specific KG to represent, manage and reuse prior knowledge extracted from historical bug reports. Then, we extract word embedding from the bug data, entity embedding and its context embedding from the bug-specific KG to enhance the initial query. The Convolutional Neural Networks (CNN) model has worked as a routine component in many IR-based bug analysis tasks because of its high performance with less need of engineered features and ability to handle multi-channel input (Zaidi et al., 2020; Jiang et al., 2020; Umer et al., 2020; Liu et al., 2019b; Xiao et al., 2019; Xie et al., 2018). Therefore, a query representation of bug embedding is finally generated by leveraging multi-level embeddings through CNN with a self-attention mechanism. The major contributions of our work are shown as follows:

- We construct a bug-specific KG, which is composed of a large number of bug-specific entities and relations extracted from the historical bug data. Based on the bug-specific KG, the entity embedding and its context embedding for the bug query can be obtained, which provides rich semantic information for the initial query.
- We combine the CNN model and self-attention mechanism to fuse multi-level embeddings, i.e., word embedding from

the original bug data, entity embedding and its context embedding from bug-specific KG, and generate the final reformulated bug query in the form of bug embedding. This informative bug embedding can be utilized as an input for existing IR-based bug analysis tasks.

- We conduct experiments on a typical IR-based bug analysis task, i.e., duplicate bug report detection. Evaluation results show that *KABR* is more effective to reformulate the bug query by comparing with the state-of-the-art approaches.

The rest of this paper is organized as follows. Section 2 introduces the background and related work. We present our approach to reformulate bug reports in Section 3. Section 4 shows our empirical study. We discuss threats to validity in Section 5. Finally, Section 6 concludes this paper and discusses some future work.

2. Background and related work

In this section, we present the background related to our approach. First, we elaborate the duplicate bug report detection task implemented in the experiment. Secondly, we introduce several query reformulation technologies used in the IR-based bug analysis tasks. Then, we describe the basic concepts and key technologies of knowledge graph. Finally, we briefly introduce the deep-learning models used in our approach, i.e., CNN and the self-attention mechanism.

2.1. Duplicate bug report detection

Duplicate bug report detection is a typical IR-based bug analysis task. Duplicate bug reports refer to different bug reports submitted for the same bug that are submitted by multiple users or because of different versions of a software project. When a new bug report is assigned, developers must first confirm whether it is a duplicate bug report, and if so, it is marked as "Duplicate" and is no longer processed. The new bug report (i.e., query) can be paired with all the historical bug reports and a trained prediction model is used to predict whether these pairs are duplicates.

Almost all the existing approaches for automatically detecting duplicate bug reports are based on text similarity. On the one hand, researchers continue to introduce new NLP (natural language processing) models for better language understanding (Xie et al., 2018; Nguyen et al., 2012; Ebrahimi et al., 2019; Liu et al., 2019; Neysiani et al., 2020; Budhiraja et al., 2018; Lazar et al., 2014). These models varied from simple TF-IDF based techniques to more complex machine learning-based methods. He et al. (2020) applied a Dual-Channel Convolutional Neural Networks (DC-CNN) model to combine the two single-channel matrices of two bug reports into a dual-channel matrix to represent a bug report pair together. More association features between bug reports could be captured to improve the accuracy of text similarity detection.

On the other hand, researchers try to enhance duplicate detection by adding extra information other than text (i.e., the title and description) as additional features. The extra information mainly comes from the categorical bug information in bug reports (Lin et al., 2016; Hindle et al., 2016; Xiao et al., 2020; Rakha et al., 2018). Xiao et al. (2020) extracted five kinds of categorical information (i.e., product, component, version, severity, and priority) and also the bug text as six types of nodes to form a heterogeneous information network (HIN) for duplicate detection. Domain information extracted from software documents, concise summaries of original bug reports, and information from Wikipedia are also used to help developers find duplicate bug reports effectively (Rastkar et al., 2014; Aggarwal et al., 2017; Hindle et al., 2016; Liu et al., 2013).

However, the above text similarity based approaches for duplicate detection have a common premise, that is, there are language commonalities between bug reports corresponding to the same bug (Chaparro et al., 2016). In practice, duplicate bug reports are often described with different words since different reporters have different writing habits. Observe the two pairs of bug reports shown in Table 1. The first bug report pair (i.e., Bug 108670 and Bug 108746) has neat language, detailed description of bug symptoms and large amount of information. In the second pair (i.e., Bug 767056 and Bug 794399), although the description section is also written in the formats of *Steps to reproduce*, *Expected Results* and *Actual Results*, the text is short and too much structured information is mixed. The quality is much worse. No matter the quality is high or low, these two pairs of bug reports share a few words (i.e., language commonalities), but they are indeed duplicate bug reports. We use the DC-CNN model to detect these two bug report pairs like He et al.'s work (He et al., 2020) which is one of the state-of-the-art approaches, but failing to find latent semantic associations between bug reports beyond language commonalities, and they are judged to be non-duplicate due to the vocabulary mismatch problem. However in Fig. 1, we can find that Bug 108670 and Bug 108746 are closely connected through entities. Therefore, whether two bugs are relevant can be judged not only by the similarity of the word-level information in bug reports, but also by the latent semantic relation between the knowledge-level information.

2.2. Query reformulation in the IR-based bug analysis tasks

As described in Section 2.1, IR-based bug analysis tasks often suffer from the uneven query quality problem, which is a longstanding issue (Chaparro et al., 2016; Haiduc et al., 2013). Researchers try to make some meaningful modifications to the initial query content (i.e., title and description) to improve the quality of bug report for better retrieval results, that is query reformulation. Sisman and Kak (2013) first introduce query reformulation in the context of IR-based bug localization. Query reformulation in the IR-based bug analysis tasks is mainly divided into two strategies (Lu and Keefer, 1994):

- *Query reduction*, which filters noisy terms (i.e., words that do not contribute to the main intent of the query) from bug reports and selects important terms, aiming to purify query information (Kim and Lee, 2020, 2019; Chaparro et al., 2016, 2019b; Rahman and Roy, 2017; Rahman et al., 2019). Haiduc et al. (2013) discarded non-noun terms or those appearing in more than 25% of the code documents, since their discriminatory power is likely to be low. Kim and Lee (2020) took into account the four objectives (i.e., query quality properties, important terms, initial information and query length), and adopted 15 objective functions to get a final subquery. Chaparro et al. (2019a) found that the information contained in the OB (i.e., the observed behavior) may result in better retrieval of bug analysis tasks. They reduced the initial query to its OB and/or BT (i.e., bug title) only.
- *Query expansion*, which focuses on important terms and adds new related terms, aiming to enrich query information. Researchers use different methods to select additional terms. Quite a lot of work on code search or bug location use different techniques (e.g., co-occurrences or TF-IDF) to order and extract important terms from relevant source code based on the pseudo-relevance feedback to complementary a query (Zhang et al., 2019b; Liu et al., 2019a; Rahman et al., 2019; Rahman and Roy, 2017; Haiduc et al., 2013). Rahman and Roy (2017), Rahman et al. (2019), Rahman and Roy (2018a) employed term co-occurrences,

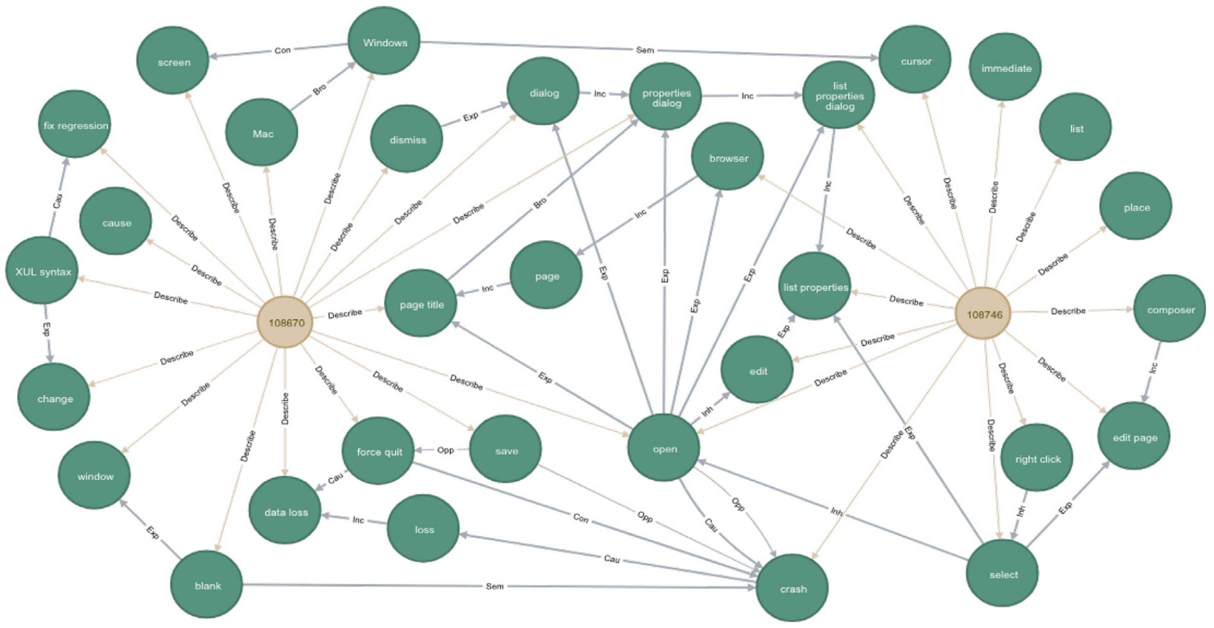


Fig. 1. The illustration of a pair of duplicate bug reports connected through text entities in bug-specific KG.

syntactic dependencies and graph-based term weighting to identify the most important keywords to build a query. Other query expansion approaches obtain additional information from some software literature. Nie et al. (2017) identified software-specific words from the pseudo-relevance feedback question and answer pairs on Stack Overflow. Lu et al. (2015) expanded the original query by replacing the words of the query with their corresponding synonyms with the same POS (parts-of-speech) in WordNet.

However, for the above two strategies, researchers are still limited to using word-level technology to determine the important information in the bug report. Our approach explores information at the granularity of entities rather than keywords. Furthermore, the existing query expansion work mainly replaces the initial query term with new terms or inserts the new terms directly into the original query. We integrate the additional information with the initial query in an embedding way to maintain the initial query syntactic properties.

2.3. Knowledge graph

A knowledge graph (KG) is a structured knowledge base originating from the semantic network that represents objective concepts/entities and their relations in the form of graph, which is one of the fundamental technologies for intelligent services such as semantic retrieval, intelligent answering, decision support, etc. (Fensel et al., 2020; Xu et al., 2017). In recent years, KG has been successfully applied to the software engineering domain, such as API recommendation (Li et al., 2018; Wang et al., 2019), software resource reuse (Lin et al., 2017; Sabou et al., 2018; Wang et al., 2017a), Git repository management (Zhao et al., 2019b), etc.

Existing IR technology mainly treats a new bug report as text and uses natural language similarity measures to calculate the similarity between the query and candidate set. The textual description in the bug report is directly processed to be a sequence of words, which ignores important semantic information. As shown in Table 1, phrases such as "Right-click" should be treated as a whole unit of knowledge, preserving the semantic structure within it. In Bug 767056, the related bug ID appears many times, such as Bug 734082, which should be recognized as a meaningful phrase rather than a series of numbers. At the same time, there is a causal relation between phrases "XUL syntax" and "fix regressions". This causal relation cannot be simply treated as a syntactic relation in a sentence. In order to better extract bug knowledge, and to explore the semantic and structural connections between these knowledge, it is necessary to establish a bug-specific knowledge graph.

2.3.1. Basic structure

As shown in Fig. 1, the bug-specific KG stores knowledge in the form of entity-relation-entity triples (h, r, t), in which h, r and t represent the head, relation and tail of a triple, respectively.

(1) **Entity:** The bug-specific entity refers to the unit cell that is distinguishable, identifiable, and has a certain semantic relation with each other in the bug data. We create the bug-specific KG based on the historical bug data. Each bug report with its textual description can be extracted as an entity. There are two types of entities with their attributes listed as follows:

Type 1: **Bug** = {*Identifier, Product, Platform, Component, Priority, Status, Reported data, Reporter, Modified data, Assignee, Version, Target, Title, Description*}

Type 2: **Text** ={*Identifier, Category*}

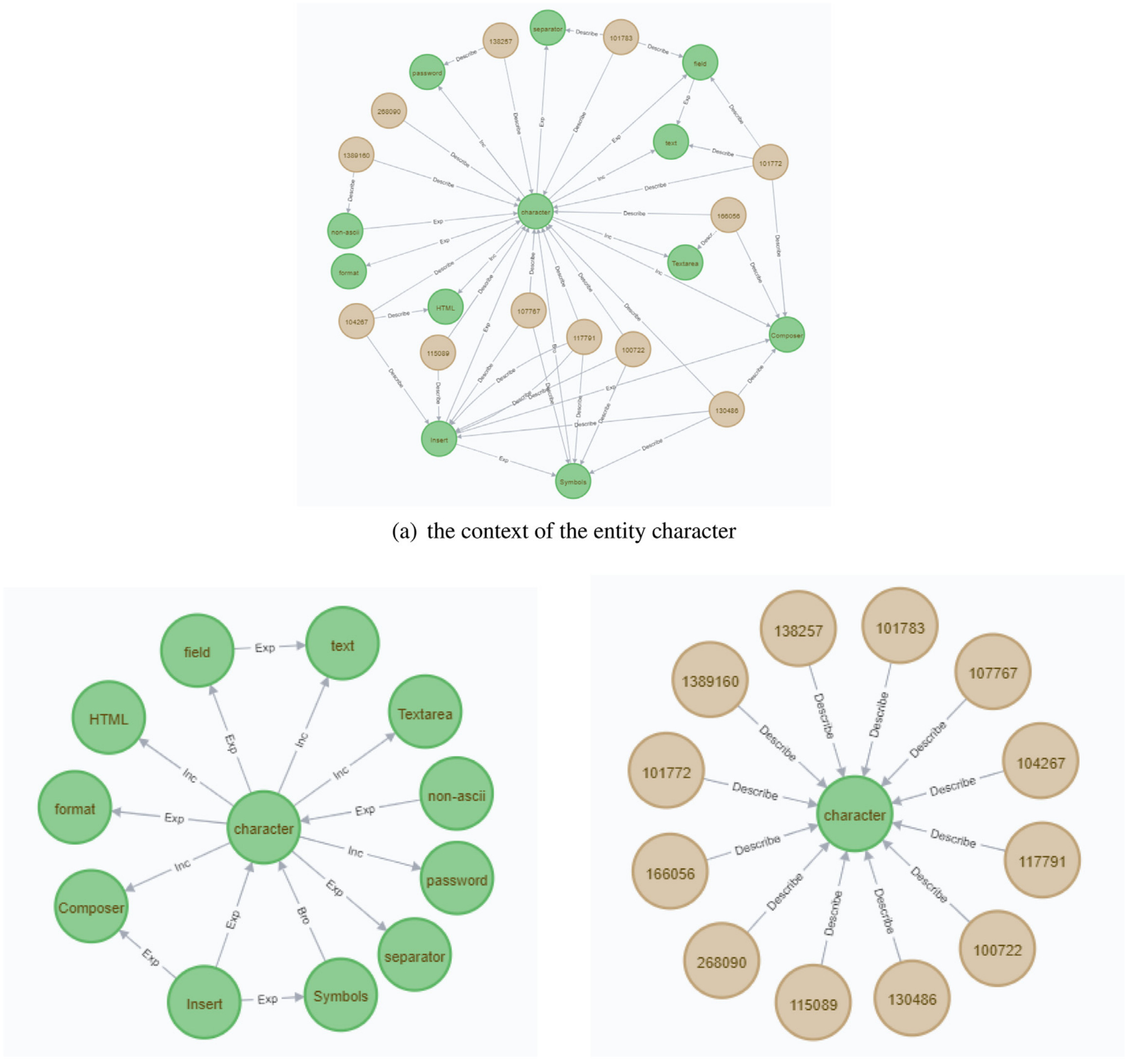
The bold words and phrases in Table 1 are entities visualized in Fig. 1. For example, "108670" is a bug entity and "force quit" is a composite text entity with an internal structure composed of two tokens. According to our previous work (Zhou et al., 2020), text entities are divided into 16 categories (i.e., *core, GUI, Network, I/O, Driver, File System, Hardware, Language, API, Standard, Platform, Framework, defect test, common adjective, common verb, and Mobile*).

(2) **Relation:** According to the type of bug-specific entities, the relations between entities are divided into three types:

Type 1: **Relation between bug entities**, which refers to the relation between two bug entities (i.e., *duplicate, dependent on, block*).

Type 2: **Relation between bug entity and text entity**, which refers to the relation between the bug entity and text entities (i.e., *describe*).

Type 3: **Relation between text entities**, which refers to the relation between two text entities. We classify the relation between



(a) the context of the entity character

(b) adjacent text entities in context

(c) adjacent bug entities in context

Fig. 2. Illustration of context of the text entity "character" in the bug-specific KG.

text entities into 8 categories (i.e., *Brother, Cause, Consensus, Opposition, Inheritance, Explanation, Inclusion, and Semantic*) referring to Chen et al.'s work (Chen et al., 2019).

(3) **Context:** The bug-specific KG contains a large number of implicit connections between bugs in the form of triples, and information extracted from a single entity is limited. We need to consider additional contextual information of an entity. Context is defined as a sub-graph centered on the target entity, consisting of all its nearest immediate neighbors. These adjacent entities represent the contextual information of the target text entity in bug reports. For example, Fig. 2(a) shows a *context* of the entity "character", which is a sub-graph centered on the text entity "character".

2.3.2. Key technologies

We construct a bug-specific KG through information extraction (IE), and introduce prior knowledge in the KG into new bug reports through entity linking and knowledge graph embedding.

(1) **Information extraction:** Named-entity recognition (NER) and relation extraction (RE) are two subtasks of information extraction. On the one hand, NER seeks to locate and classify named entities in unstructured text into predefined categories (McCallum and Li, 2003). On the other hand, relation extraction (RE) aims to classify the semantic relation between two entities in a sentence (Zhou et al., 2016).

(2) **Entity linking:** Given a knowledge graph containing a set of entities E and a text collection in which a set of named entity mentions M are identified in advance, an entity linking (EL) system aims to map each textual entity mention $m \in M$ to its corresponding entity $e \in E$ in KG (Dalton et al., 2014; Liu and Fang, 2015; Shen et al., 2015). If some entity mention in text does not have its corresponding entity record in the given knowledge graph, they are called unlinkable mentions and labeled as " **NIL** ". The EL system mainly exploits two types of information: the similarity of the mention to the candidate entity string, and coherence between the candidate entity and other entities in the text.

(3) **Knowledge graph embedding:** Under the premise of preserving the structural information of the original knowledge graph, knowledge graph embedding aims to learn a low-dimensional representation vector for each entity and relation for calculation and reasoning. Since its introduction in 2013, the Translation-based knowledge graph Embedding (TransE) model (Bordes et al., 2013) has been widely used as the basis for knowledge base vectorization, solving multi-relational data processing problem. TransE wants $\mathbf{h} + \mathbf{r} \approx \mathbf{t}$ when (h, r, t) holds, where $\mathbf{h}$, $\mathbf{r}$

and $\mathbf{t}$ are the corresponding representation vectors of h, r and t. Therefore, TransE assumes the score function:

$$f_r(h,t) = \|\mathbf{h}+\mathbf{r}-\mathbf{t}\|_2^2$$

To encourage the discrimination between correct triples and incorrect triples, the following margin based ranking loss is used for training:

$$L = \sum_{(h,r,t)\in S}\sum_{(h',r,t')\in S'} max(0, f_r(h,t)+\gamma - f_r(h',t'))$$

where γ is the margin, S and S' are the set of correct triples and incorrect triples.

The bug-specific KG aggregates a large amount of historical bug knowledge by reducing the data granularity from the document level to the entity level, providing an accessible external information source for query enrichment.

2.4. Convolutional neural networks

The Convolutional Neural Networks (CNN) model is a popular architecture for dealing with NLP tasks and has achieved remarkable results in bug analysis tasks (Zaidi et al., 2020; Umer et al., 2020; Liu et al., 2019b; Xiao et al., 2019; Xie et al., 2018; Lee et al., 2017). CNN perform excellently in extracting n-gram features by utilizing convolutional filters to capture local correlations in a parallel way.

In 2018, Wang et al. (2018) proposed a novel framework called KCNN (knowledge-aware convolutional neural networks) for news recommendation. With its multi-channel input, CNN are used to generate a embedding vector for each piece of news by associating each word in the news content with a relevant entity in the knowledge graph. Over the years, a lot of work has combined prior knowledge with CNN to improve performance (Hou et al., 2021; Chen et al., 2020; Zhang et al., 2018; Li et al., 2020; Gao et al., 2018). This method is also suitable for relevant bug file retrieval. CNN can fully exploit the internal features of the query content, and inject additional knowledge to enrich query information for better bug understanding.

In this subsection, we introduce the basic Kim-CNN architecture (Kim, 2014) used in our approach.

Let $w_{1:n}$ be the raw input of a sentence of length n, and $w_{1:n} = [w_1 w_2 \cdots w_n] \in \mathbb{R}^{d\times n}$ be the word embedding matrix of the input sentence, where $\mathbf{w}_i \in \mathbb{R}^{d\times 1}$ is the embedding of the ith word in the sentence and d is the dimension of word embedding. A convolution operation with filter $h \in \mathbb{R}^{d\times l}$ is then applied to the word embedding matrix $\mathbf{w}_{1:n}$, where $l(l \leqslant n)$ is the window size of the filter. Specifically, a feature c_i is generated from a sub-matrix $\mathbf{w}_{i:i+l-1}$ by

$$c_i^h = f(h * \mathbf{w}_{i:i+l-1} + b)$$

where f is a non-linear function, $*$ is the convolution operator, and $b \in \mathbb{R}$ is a bias. After applying the filter to every possible position in the word embedding matrix, a feature map $\mathbf{c^h} = [c_1^h, c_2^h, \ldots, c_{n-l+1}^h]$ is obtained. Then, a max-overtime pooling operation is used on feature map c to identify the most significant feature:

$$\tilde{c}^h = max\left\{\mathbf{c}^h\right\} = max\left\{c_1^h, c_2^h, \ldots, c_{n-l+1}^h\right\}$$

One can use multiple filters (with varying window sizes) to obtain multiple features, and these features are concatenated together to form the final sentence representation. In IR-based bug analysis tasks, the vector distance between query embeddings will be further calculated to measure the similarity between bugs.

2.5. Self-attention

Because CNN only consider sequential n-grams that are consecutive on the surface string, it cannot grasp long term contextual information and correlation between non-consecutive words, while this kind of correlation plays an important role in bug understanding. Let us focus on the first sentence in the description section of Bug 108670. There is causality between entity "force quit" and entity "data loss", and negation between entity "force quit" and entity "dismiss". Worse, like most hybrid frameworks, CNN treat all words equally and neglect the fact that different words have different contributions to the semantics of a text. However, as discussed in Section 2.2, researchers have been trying to select important terms to reformulate query.

The attention mechanism aims to filter out a small amount of important information from a large amount of source data by weighting, and focuses on these important information. In 2017, Google published the self-attention model (Vaswani et al., 2017). Self-attention is a special case of attention mechanism that only requires a single sequence to compute its representation. At present, it has been successfully applied to a variety of NLP tasks based on neural network models such as RNN/CNN/Bert (Zhang et al., 2019a; Li et al., 2020; Liu et al., 2020). However, as far as we know, attention mechanism has not been introduced into bug reformulation tasks.

In this paper, in contrast to other bug expansion methods, we try to enrich the initial query by introducing structured knowledge facts from the bug-specific KG in the form of informative entities for better bug understanding. Before information fusion, we combine the self-attention mechanism with CNN to give different weights to the input multi-level embeddings, so as to capture important words/entities in sentences and syntactic dependencies between non-consecutive words. The specific implementation steps will be introduced in Section 3.3.1.

3. Approach

Fig. 3 shows the overall framework of *KABR* for knowledge-aware bug report reformulation by leveraging multi-level embeddings. It mainly consists of three steps: bug-specific knowledge graph construction, multi-level embedding and query generation. In this section, we present details of these three steps.

3.1. Bug-specific knowledge graph construction

According to the study of Zhou et al. (2020), the category of most bug-specific named entities is determined by the type of the component to which the bug belongs. Hence, in order to improve the representativeness of prior data, we collected 10,000 verified fixed bug reports by stratified sampling based on components from Mozilla's BTS. Details of bug data are described in Section 4.2. We collected the title, description and categorical information of each bug report to form the bug-specific KG. The construction of KG is mainly divided into three parts: knowledge extraction, entity alignment, and knowledge enrichment.

3.1.1. Knowledge extraction

The bug-specific KG is defined as $G = (E, R)$ with entities E as nodes and relations R as edges. Bug-specific entities are divided into two types: *bug* and *text*. On the one hand, each bug report is a bug entity, and categorical information in the bug report contains the corresponding attributes. The bug entity takes the unique bug ID as its value, such as "108670". We also extract the relation between bug reports from categorical information (i.e., *duplicate, dependent on, and block*) as relations between bug entities. On the other hand, we apply the *DBNER* method (Zhou et al., 2020) to recognize text entities from titles and descriptions. In addition, we use attention-based Bidirectional Long Short-Term Memory Networks (Att-BiLSTM) to capture relations between text entities in bug textual description (Zhou et al., 2016). There is a *describe*

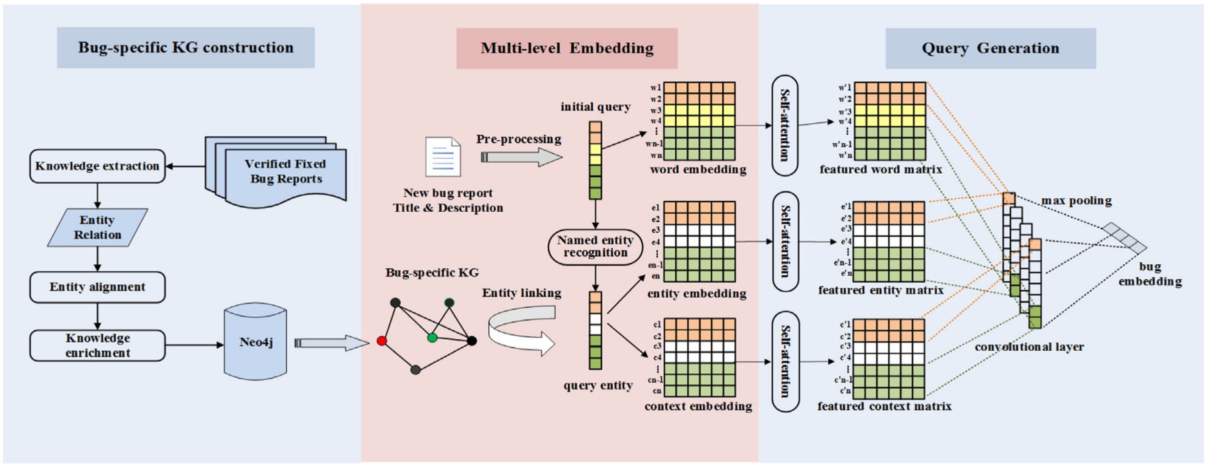


Fig. 3. The overall framework of knowledge-aware bug report reformulation leveraging multi-level embeddings.

relation between each bug entity and the text entity identified from its textual description. Taking Bug 108670 as an example, we extract 17 text entities and 12 relations between text entities from its title and description. Finally, we obtain a total of 52,680 entities and 184,167 entity-relation-entity triples (h, r, t) by analyzing the 10,000 verified fixed bug reports. The Neo4j[4] (a NOSQL graph database) is used to store and manage these triples.

3.1.2. *Entity alignment*

Historical bug reports are submitted by different users or testers. Different reporters may describe the same issue in different ways. The purpose of entity alignment is to determine whether two text entities point to the same physical object and then merge the same text entities. The content of the bug report usually does not follow standard naming rules and descriptive formats. We first pre-process the name (i.e. text entity value) with the Natural Language Toolkit (NLTK).[5] We use the Snowball (a library in NLTK) stemmer to transform the words into their root forms (e.g., "caused" and "causes" are reduced to "cause") in order to unify similar words into a common representation. We use the Truecase (a library in NLTK) to restore uppercase to the most probable state, for example, general terms such as "Page Title" are reduced to "page title", while software domain terminology such as the operating system "Windows" remains "Windows" instead of the operating interface "windows". The text entity "Windows" belongs to the *Platform* category, while the text entity "window" belongs to the *GUI* category. They cannot be aligned as the same text entity. Considering that our data comes from the same software project, we simply match the similarity of text entity values and categories to implement text entity alignment.

3.1.3. *Knowledge enrichment*

Naturally, the larger the KG is, the higher the possibility of providing more relevant prior knowledge for query. However, the scale updating of KG is a continuous and long work. In this paper, we only discuss the benefits of a finite scale bug-specific KG to the new bug analysis task. Density (number of KG edges) of candidate entity affects entity linking (EL) performance. The historical bug reports that make up the bug-specific KG are randomly sampled, and their quality is uneven. As shown in Table 1, the quality of Bug 767056 is poor, and the entity-relation-entity triples (h, r, t) extracted from its description content are relatively sparse and lack of diversity. At the same time, a bug with poor quality is easy to form an island, that is, the entities that constitute the bug has no or weak relations with other entities.

In order to improve the accuracy of entity linking between the query entity and the KG, we further enrich the bug-specific KG density in two ways. On the one hand, we obtain a batch of synonyms of each text entities from ConceptNet[6] (a knowledge base with rich lexical relations), and select the top 3 weight synonyms that are most similar. If these synonyms do not appear in the bug-specific KG, they will be updated as new text entities. For example, if the synonym "break down" is obtained according to the target text entity "crash", a triple "crash, *Consensus*, break down" is updated. The category of "crash" is *common verb*, and the new text entity "break down" is also set to the same category *common verb* . On the other hand, we get some co-occurring words and phrases for each text entity from the original bug reports in BTS. We refine the co-occurrences with the Pointwise Mutual Information (PMI) measure (Church and Hanks, 1989) which has been widely used (Damani, 2013). We only choose co-occurring words and phrases from top 5 results ordered by PMI values. Similar to the process of synonyms above, if co-occurrence words do not appear in the bug-specific KG, they will be updated as new text entities with the same category of target text entity. The difference is that the relation category between entities is set as *Semantic*.

3.2. *Multi-level embedding*

Based on the bug-specific KG, we can automatically extract multi-level features to reformulate the initial query (i.e., title and description in the new bug report). We learn word embedding from the original bug data, entity embedding and context embedding from the bug-specific KG to form an informative query embedding.

3.2.1. *Word embedding*

In a new bug report, there are only a simple title and description with a few categorical bug attributes. For each bug report, we combine its title and description into a single document as the initial query (Q). If the product and component of the new bug are also provided, we attach the product and component to the end of the document (Q). Then, we pre-process Q with the Natural Language Toolkit (NLTK). After tokenization, Q is divided

[4] https://neo4j.com/
[5] http://www.nltk.org/
[6] http://www.conceptnet.io/

into a token sequence denoted as $Q = \{w_1, w_2, \dots, w_n\}$, where n is the length of the document.

We pre-train word embedding for each token using word2vec[7] model on the original bug reports in BTS. The word embedding matrix of the query is denoted as $Q_{\mathbf{w}} = [\mathbf{w_1 w_2 \cdots w_n}] \in \mathbb{R}^{d\times n}$ where d is the dimension of word embedding.

3.2.2. Entity embedding

An entity is the smallest unit of knowledge and contains rich semantic information. We utilize the TransE (Bordes et al., 2013) model (as introduced in Section 2.3.2) to learn knowledge graph embedding for each entity and relation in the KG. Prior knowledge from the bug-specific KG can be injected into the initial query through entity linking. We first utilize the *DBNER* (Zhou et al., 2020) method to recognize text entities in the query. Then we select and rank candidate entities by calculating similarity between the entity mention m in the query and the text entity e existing in the bug-specific KG. We consider three similarity features:

- *Name String Comparison*: Because the entity alignment has been done in earlier stage, there is no case of different entities with the same name in the bug-specific KG. Therefore, if some name n equals m, that is, exact matching, then the entity e with the name n is the only candidate entity. Since queries come from different software projects (including but not limited to Mozilla), we also use partial matching (i.e., *the entity name n is wholly contained in or contains the entity mention m*) to ensure a high recall rate.
- *Textual Context*: We use cosine similarity to calculate the text similarity between two entities.
- *Entity Category*: This feature is to indicate whether the category of the entity mention in text is consistent with the category of the candidate entity in the KG.

In general, the similarity between the entity mention m and candidate entity e is a linear weighted fusion of the above three features as follows:

$$S_{sim}(m,e) = \lambda_1 \times S_{string} + \lambda_2 \times S_{COS} + \lambda_3 \times S_{category}$$

where λ_1, λ_2, λ_3 are three similarity parameters, and meet $\lambda_1 + \lambda_2 + \lambda_3 = 1$.

We selected 400 bug reports from Mozilla and manually annotated them as the entity linking data set. A total of 2,300 entities were selected as the entity linking targets of the KG. In experiments, we use the control variable method, take the recall rate as the evaluation metric to adjust λ_1, λ_2 and λ_3 to obtain the optimal setting. In order to ensure the accuracy, we adopt the 5-fold cross validation. We find that reducing the weight of λ_3 (i.e., entity category) will significantly increase the recall rate, while increasing the parameter λ_2 (i.e., cosine similarity) will improve the recall rate, so λ_3 is relatively small. When parameter λ_1 is increased, the recall rate is also improved to a certain extent. Considering that string matching has more advantages over word-level entity linking, the relative proportion of λ_1 is larger. Finally, the parameters are determined as $\lambda_1 = 0.52$, $\lambda_2 = 0.37$, $\lambda_3 = 0.11$ by comparison.

After entity linking, each token in the query w_i learns a corresponding entity embedding $\mathbf{e_i} \in \mathbb{R}^{k\times 1}$. Non-entities and unlinkable entities are set as zero. The entity embedding matrix of the query is denoted as $Q_{\mathbf{e}} = [\mathbf{e_1 e_2 \cdots e_n}] \in \mathbb{R}^{k\times n}$, where k is the dimension of entity embedding.

7 https://radimrehurek.com/gensim/models/word2vec.html

3.2.3. Context embedding

Fig. 2(a) shows the whole context (i.e., the nearest immediate neighbors) of the entity "character" in the bug-specific KG. Fig. 2(b) shows the adjacent text entities, and Fig. 2(c) shows the adjacent bug entities (i.e., bug reports that described by the target entity). The amount of information in a bug entity is richer than that of a text entity because it contains many attributes. The latent connections between the new bug and historical bugs can be enhanced by the bug-text-text-bug path, as the path "*108670-force quit-crash-108746*" shown in Fig. 1. Quite a few bug reports do not share the same text entities, but are indirectly linked through triples. For this reason, we set different weights for these two types of entities to perform context embedding. Given the context of a target entity e, the context embedding is calculated as follows:

$$\mathbf{c} = \frac{1}{|context(e')|} \sum_{\mathbf{e_i} \in \mathbf{context}(\mathbf{e}')} \mathbf{e^i} + \frac{1-\alpha}{|context(e'')|} \sum_{\mathbf{e_j} \in \mathbf{context}(\mathbf{e}'')} \mathbf{e^j}$$

where $context(e')$ is the contextual bug entity, $context(e'')$ is the contextual text entity. The weight value α is detailed in Section 4.3.2. The context embedding matrix of the query is denoted as $Q_{\mathbf{c}} = [\mathbf{c_1 c_2 \cdots c_n}] \in \mathbb{R}^{k\times n}$, where k is the dimension of context embedding and is the same as the dimension of entity embedding.

The word-level feature (i.e., word embedding trained on the bug data) and knowledge-level feature (i.e., entity embedding and context embedding trained on the KG) are learned by different models, and the dimension of their vectors is different from each other. So we should transform them into the same vector space. We transform entity embedding as:

$$g(Q_{\mathbf{e}}) = [g(\mathbf{e}_1)g(\mathbf{e}_2)\cdots g(\mathbf{e}_n)]$$

Similarly, we transform context embedding as:

$$g(Q_{\mathbf{c}}) = [g(\mathbf{c}_1)g(\mathbf{c}_2)\cdots g(\mathbf{c}_n)]$$

We use a non-linear function to transform the embeddings:

$$g(\mathbf{e_i}) = tanh(Me_i + b)$$

$$g(\mathbf{c_i}) = tanh(Mc_i + b)$$

where $M \in \mathbb{R}^{d\times k}$ is the trainable transformation matrix and $b \in \mathbb{R}^{d\times 1}$ is the trainable bias.

At this point, the bug query is transformed from a sequence of tokens into three equally-sized embeddings (low-dimensional space vectors). At the same time, through named entity recognition (NER), developers can see the entities in the query. After entity linking, developers can also see the entity and its context in the bug-specific KG corresponding to the entity mention in the query.

3.3. Query generation

After multi-level embedding, we have obtained rich information from historical bug reports. We also need to fuse them to generate a uniform representation to facilitate subsequent retrieval.

3.3.1. Self-attention

Not all words in a bug query are useful for retrieval. Recently, researchers have adopted a query reduction strategy to remove noise data from the initial query and retain important terms (i.e., words) as a new query (Chaparro et al., 2019b,a). In our approach, we use the self-attention mechanism to assign higher weight to important words and entities in the query, so as to achieve the effect of noise suppression.

The word embedding matrix, transformed entity embedding matrix, and transformed context embedding matrix are fed into three self-attention layers, respectively. We pass the query matrix $Q_{\mathbf{w}}$ through the full connection and then activate it with a tanh function, to obtain a new word matrix $X_{\mathbf{w}}$:

$$X_{\mathbf{w}} = Q_{\mathbf{w}} W_{s2} tanh(W_{s1} Q_{\mathbf{w}} + b') \in \mathbb{R}^{n\times n}$$

We use the softmax function to process each row of matrix $X_{\mathbf{w}}$ to form a self-attention matrix $A_{\mathbf{w}}$:

$$a_{ij} = \frac{e^{x_{ij}}}{\sum_{k=1}^{n} e^{x_{ik}}}$$

where W_{s1}, W_{s2}, b' are the attention parameters. Then, the featured word embedding can be calculated:

$$t_{\mathbf{w}_i} = \sum_{j=1}^{n} (a_{ij}\mathbf{w}_i)$$

In the same way, we can get the featured entity embedding $t_{\mathbf{e}_i} = \sum_{j=1}^{n} (a'_{ij} g(\mathbf{e}_i))$ and the featured context embedding $t_{\mathbf{c}_i} = \sum_{j=1}^{n} (a''_{ij} g(\mathbf{c}_i))$. In this way, important words and entities are given higher weight. At the same time, attention matrix visualization can directly display the important content of query for developers.

3.3.2. Query representation

After the self-attention layer, we get three featured embedding matrices of the same size. Given the three embeddings above, a straightforward way to combine them is to treat the entities as pseudo words and concatenate them to the word sequence (Wang et al., 2017b), as

$$W = [t_{\mathbf{w}_1} t_{\mathbf{w}_2} \cdots t_{\mathbf{w}_n} t_{\mathbf{e}_1} t_{\mathbf{e}_2} \cdots t_{\mathbf{e}_n} t_{\mathbf{c}_1} t_{\mathbf{c}_2} \cdots t_{\mathbf{c}_n}]$$

Then, such query matrix W can be fed into existing matching models for similarity calculation. However, the concatenating strategy breaks the connection between words and associated entities and is unaware of their alignment. Here we align them into three colored image-like channels of the CNN model for combining word semantics and knowledge information. The query matrix W is constructed as:

$$W = [[t_{\mathbf{w}_1} t_{\mathbf{e}_1} t_{\mathbf{c}_1}][t_{\mathbf{w}_2} t_{\mathbf{e}_2} t_{\mathbf{c}_2}] \cdots [t_{\mathbf{w}_n} t_{\mathbf{e}_n} t_{\mathbf{c}_n}]] \in \mathbb{R}^{d\times n\times 3}$$

Then, we apply multiple filters $h \in \mathbb{R}^{d\times l\times 3}$ with varying window sizes l to extract specific local patterns in the bug query based on the Kim-CNN model (Kim, 2014) introduced in Section 2.3. The local activation of sub-matrix $W_{i:i+l-1}$ with respect to h can be written as:

$$c_i^h = f(h * w_{i:i+l-1} + b)$$

We use a max-overtime pooling operation on the output feature map to choose the largest feature:

$$\tilde{c}^h = max\left\{\mathbf{c}^h\right\} = max\left\{c_1^h, c_2^h, \ldots, c_{n-l+1}^h\right\}$$

Then, all features are concatenated together and taken as the final representation $\mathbf{k}(Q)$ of the input bug query q:

$$\mathbf{k}(Q) = [\tilde{c}^{h_1} \tilde{c}^{h_2} \cdots \tilde{c}^{h_m}]$$

where m is the number of filters.

After the above three steps, we fuse multi-level embeddings with CNN and the self-attention mechanism to represent the initial query as an informative bug report embedding. This embedding can be used in various IR-based bug analysis tasks to analyze the correlation between bugs.

4. Empirical study

4.1. Research questions

Our reformulation approach *KABR* is designed to inject external structured knowledge into the initial bug report (i.e., query) to enrich it and represent it as an informative bug embedding, which can be directly used as the input of the existing matching models to calculate the similarity between bug reports . To show the effectiveness of *KABR*, we conduct experiments on the duplicate bug report detection task as shown in Fig. 4. For a pair of bug reports to be detected, we first perform data pre-processing, and then reformulate them through *KABR* to enrich the information and generate a pair of bug embeddings. We mainly focus on the following three research questions:

RQ1: Do our reformulated queries perform better for duplicate bug report detection than without query reformulation ?

KABR is designed to introduce prior knowledge from the bug-specific KG to enrich new bug reports to improve bug retrieval. In addition, *KABR* leverages three embeddings for bug report reformulation. So we first investigate *KABR* to show whether it is effective for duplicate bug report detection.

RQ2: How effective is *KABR* to query with limited information?

If the information of the query is limited, fewer entities can be recognized from the query. *KABR* introduces prior knowledge into the initial query through entity linking. The number of entities in the query directly affects the amount of external information obtained. In practice, queries are new bug reports submitted by users. The new bug report is relatively short and lacks a lot of necessary information. We need to investigate whether *KABR* is still effective for the bug query with limited information.

RQ3: How effective is *KABR* compared with the existing query reformulation approaches targeting duplicate bug report detection ?

There have been a number of works on query reformulation that either expand or reduce the bug reports for a better search query. *KABR* is not only to add relevant information, but also to maintain the original sequence structure of the bug query through embedding fusion. So we need to investigate whether *KABR* is more effective over the state-of-the-art query reformulation approaches.

4.2. Data sets

The data sets include two parts: the duplicate bug report data set and the bug-specific KG data set.

4.2.1. Duplicate bug report data set

The duplicate bug report data set is constructed based on two data sets used in prior duplicate detection research. We selected 3 of 12 open source projects (i.e., Mozilla Core, Firefox, Thunderbird) from *BugRepo*[8] used by Xie et al. (2018), and selected the Eclipse and Open Office projects from the data set provided by Chaparro et al. (2019b). Each project contains more than 3,000 duplicate bug reports (also known as issues) to ensure three relatively large data sets (i.e., Mozilla, Eclipse and Open Office) for CNN. Mozilla core, Firefox and Thunderbird are all products of Mozilla project. We combined them to form the Mozilla data set. Mozilla is a web browser, Eclipse is open-source integrated development environments, and Open Office is an office software similar to Microsoft Office. The types of these three projects are different, so they are suitable for cross-projects experiments.

[8] https://github.com/LogPAI/Bugrepo

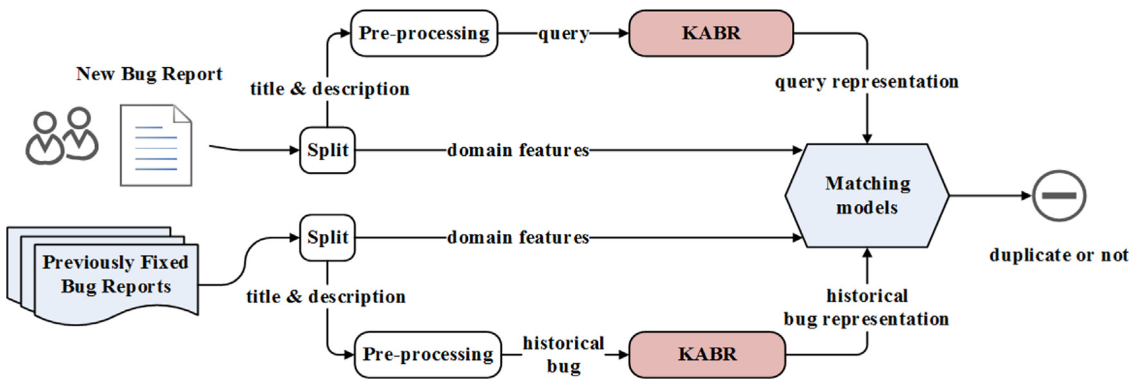


Fig. 4. The process of duplicate detection with KABR.

Table 2
Statistics of our data sets.

Project	Product	# Components	# Issues	# Duplicates	# BRS in KG
Mozilla	Mozilla Core	136	205,069	44,691	3,024
	Firefox	49	115,814	33,815	1,180
	Thunderbird	25	32,551	12,501	712
Eclipse		27	209,056	28,518	–
Open Office		12	31,138	3,019	–
Total		249	593,628	122,544	4,916

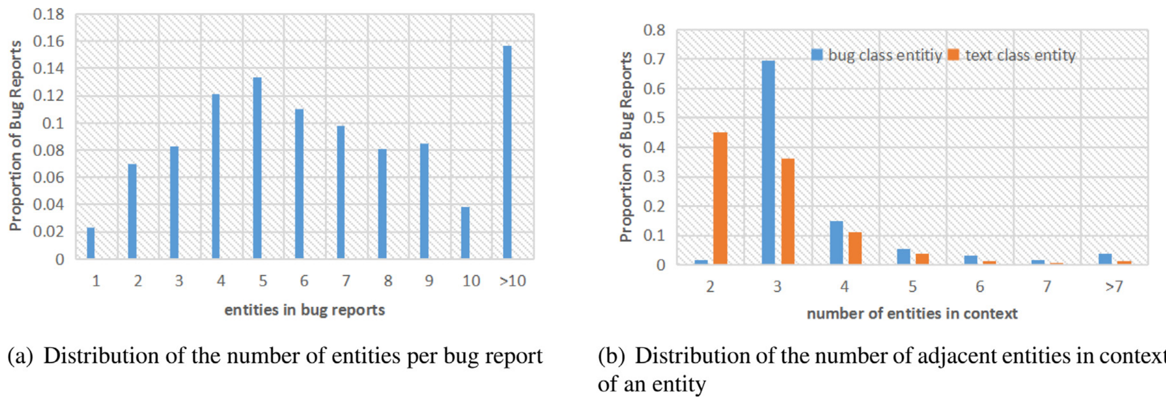


Fig. 5. The statistical distribution of data in the bug-specific KG.

4.2.2. *Bug-specific KG data set*

We totally collected 10,000 verified fixed bug reports by stratified sampling based on components from Mozilla's BTS to construct our bug-specific KG. In order to improve the coverage of text entities, we expanded the sampling rate for three experimental projects. The number of bug reports (i.e., BRS in Table 2) sampled were: 3,024 for Mozilla Core, 1,180 for Firefox, 712 for Thunderbird, and 5,084 for other products in the Mozilla project. None of the bug reports in the duplicate bug report data set appeared in the KG. In addition, in order to ensure the quality of bug-specific KG, the bug reports we collected should contain clear *Steps to Reproduce*, *Expected Results* and *Actual results* in the textual description. We employed regular expressions to automatically check the format of each sampled bug report. If the above three phrases were not detected, such as the Bug 108670 shown in Table 1, the content was determined according to manual analysis. Given that the bug has been verified as fixed, developers can easily identify the three parts needed. The first author and the fourth author conducted manual analysis together. For the bug report that really lacks necessary elements, we repeatedly sample on the same component until it meet the requirements. We also sort out the statistical distribution of data in the bug-specific KG, as shown in Fig. 5.

4.3. *Experiment setup*

4.3.1. *Baseline*

To study the effectiveness of our bug report reformulation approach, we set different baselines for different RQs:

(1) For RQ1, we used the initial query (i.e, title and description) as the baseline query (a.k.a. Q_{IN}).

(2) For RQ2, in order to simulate new bug reports with limited information, we only extracted the title from the bug report as the baseline query. After all, each bug report must contain a title.

In view of the relevant work of Chaparro et al. (2019b), they believe that employing bug reduction strategy can improve the duplicate bug report detection task, that is, select the OB (i.e., observed behavior) and/or BT (i.e., bug title) and remove the rest of the textual bug description. Through experiments, they find that the combination of OB and BT is the best. In order to evaluate *KABR* more comprehensively, we also set up three query reduction strategies proposed by Chaparro et al. as baselines, including observed behavior only (a.k.a. Q_{OB}), bug title only (a.k.a. Q_{BT}), observed behavior and bug title (a.k.a. Q_{OB-BT}).

(3) For RQ3, we applied three query expansion strategies. A variety of bug report expansion approaches have been proposed

in the field of TR. We found, however, that not all these were applicable to our circumstances (i.e., corpora based on bug text rather than source code). Therefore, we adjusted *BLIZZARD*, a new approach proposed by Rahman and Roy (2018a) to be more suitable for the duplicated bug report detection task and serve as a baseline.

- **Synonym expansion**: we added the top 3 synonyms of each token from the open semantic network ConceptNet to expand the query (a.k.a. Q_{SE}).
- **Entity expansion**: we returned adjacent text entities in the context of each entity from the bug-specific KG through entity linking to expand the query (a.k.a. Q_{EE}).
- **Keyword expansion**: referring to *BLIZZARD*, we first collected Top-10 historical bug reports from the bug-specific KG data set retrieved by a query, and developed a text graph using co-occurrences and syntactic dependencies among the words from each bug report. Then we applied PageRank to the text graph for identifying important keywords. At last we combined the query with the highly Top-K ($8 \leqslant K \leqslant 30$) weighted keywords as the reformulated query (a.k.a. Q_{KE}).

4.3.2. Training details

In the experiment, we use C4.5 and softmax classifier as the matching model to process the output of *KABR* (i.e., the bug embedding). For the baseline queries, we use C4.5 and DBR-CNN as the matching model to process the input query text.

- **C4.5** (Aggarwal et al., 2017), is a state-of-the-art feature-based machine learning model for duplicate bug report detection (Hindle et al., 2016). We consider textual and categorical features in our study.
- **Softmax classifier** (Kim, 2014), is a common linear classifier, which is often used as the output layer of various neural network models including CNN.
- **DBR-CNN** (Xie et al., 2018), is a deep structured semantic model for duplicate bug report detection, which combines traditional CNN model (i.e., CNN with a softmax layer) with categorical features.

For *KABR with C4.5* (a.k.a. *KABR-C*), *KABR with softmax classifier* (a.k.a. *KABR-S*) and *DBR-CNN*, we pretrained 100-dimensional word embedding on the three bug data repositories (i.e., Mozilla, Eclipse and Open Office), and fine-tuned them during the learning process. We selected 20% of each duplicate bug report data set according to the chronological order as the duplicate bug report test set (a.k.a. $test_{IN}$). Then, the 5-fold cross-validation method is used for the remaining 80% of each data set. We randomly divided the remaining bug reports into five copies, and four of them were selected in turn as the training set and the other as the validation set. For *C4.5* only, we also carried out a 5-fold cross-validation. We extract 80% of the data set as the training set and 20% of the data set as the test set.

For RQ2, we separate the bug titles from the test set $test_{IN}$ as a new test set (a.k.a. $test_{BT}$) . We further reconstruct new data sets according to different baseline query strategies for RQ2 and RQ3, and train and test the classifier based on the new data sets.

4.3.3. Parameter settings

We chose TransE (Bordes et al., 2013) to process the knowledge graph and learn entity embedding and context embedding, and used the non-linear transformation function in the query generation step. We utilized the following parameters: 100-dimension word embedding; 100-dimension entity embedding; filter number = 100; filter length = 5; weight $\alpha = 0.4$. We used Kingma and Ba (2015) to train *KBRA* by optimizing the log loss.

As shown in Fig. 6(a), when both the word embedding dimension d and the entity embedding dimension k are set to 100, the F1 value reaches the highest for Mozilla. We explored the value of the weight α ($\alpha \in [0, 1]$) in context embedding $\mathbf{c}$. As shown in Fig. 6(b), the F1-measure increases as the value of weight α gets larger, reaches a maximum near 0.4, and then slowly decreases until it is steady. Such a trend of change shows that in context embedding, properly increasing the weight of the bug entity can help extract more contextual information.

4.3.4. Evaluation metrics

Two widely used metrics *F1-measure* (Xie et al., 2018) and *Accuracy* (Aggarwal et al., 2017) are employed to measure the performance of duplicate detection. The higher the F1-measure and Accuracy are, the better the detection performance presents. In our study, we also used these two metrics to measure the effectiveness of our approach.

F1-measure is the harmonic mean of precision (P) and recall (R). Precision (P) measures the ratio of the number of positive samples correctly classified as duplicate to the number of samples classified as duplicate. Recall (R) measures the ratio of the number of positive samples correctly classified as duplicate to the number of positive samples. F1 is defined as follows:

$$F1 = \frac{2 \times P \times R}{P + R}$$

The *Accuracy* is defined as:

$$Accuracy = \frac{Number\ of\ correct\ detections}{Number\ of\ bug\ report\ samples}$$

4.4. Experiment results

4.4.1. RQ1: Effectiveness of KABR

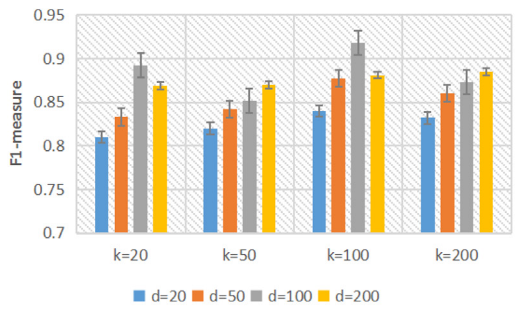
To verify whether our reformulation approach can improve the IR performance, we use the initial query Q_{IN} as the baseline.

(1) *Comparison between different matching models.* Table 3 shows the comparative results among different comparative models. We first compare the duplicate bug detection effectiveness with or without *KABR*. When we use *KABR* to reformulate the initial query, *KABR-C* improves *C4.5* by 11.4% for F1-measure and 8.5% for accuracy on Mozilla, 8.8% for F1-measure and 4.8% for accuracy on Eclipse, 7.8% for F1-measure and 10% for accuracy on Open Office. Combined with *KABR*, the effectiveness of the original approach is improved. Then, we compare *KABR-S* with *DBR-CNN*, which also generates the bug report representation via CNN model. *KABR-S* improves *DBR-CNN* by 11.9% for F1-measure and 7.0% for accuracy on Mozilla, 7.4% for F1-measure and 8.0% for accuracy on Eclipse, 12.2% for F1-measure and 12.6% for accuracy on Open Office. So we can conclude that *KABR* does introduce useful knowledge from the bug-specific KG into the initial query and improves the retrieval effectiveness.

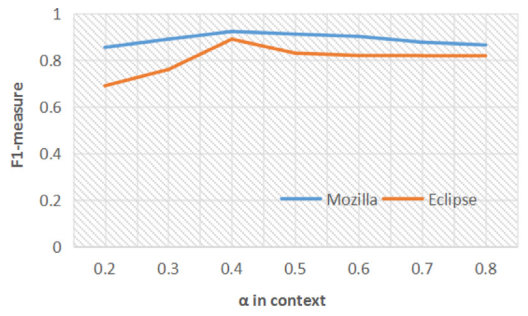
In order to further investigate the reasons for the increase, we compare the effectiveness with or without the self-attention mechanism. On Mozilla, with attention mechanism, the F1-measure increases by 3.0% and the accuracy increases by 3.3%. On Eclipse, with attention mechanism, the F1-measure increases by 2.8% and the accuracy increases by 3.1%. On Open Office, with attention mechanism, the F1-measure increases by 2.4% and the accuracy increases by 1.8%. The comparison confirms that the introduction of attention mechanism is beneficial to the model.

(2) *Comparison between different embeddings.*

Since *KABR* combines three embeddings (i.e., word embedding from the bug data, entity and context embeddings from bug-specific KG), we want to investigate the detection effectiveness of different combinations of the three embeddings. From Table 4,



(a) Impact of different dimensions of entity embedding k and dimensions of word embedding d



(b) Impact of different weight α of context embedding c

Fig. 6. Performance of knowledge-aware bug report reformulation under different parameter settings.

Table 3
Results of different bug retrieval models.

Approach	Mozilla		Eclipse		Open Office	
	F1	Accuracy	F1	Accuracy	F1	Accuracy
C4.5	0.827	0.845	0.822	0.839	0.862	0.837
KABR-C	0.921	0.917	0.895	0.882	**0.929**	**0.92**
DBR-CNN	0.846	0.875	0.852	0.863	0.823	0.809
KABR-S	**0.94**	**0.936**	**0.915**	**0.932**	0.924	0.911
KABR-S without attention	0.912	0.906	0.890	0.904	0.902	0.895

Table 4
Results of KABR-S with different embeddings.

Approach	Mozilla		Eclipse		Open Office	
	F1	Accuracy	F1	Accuracy	F1	Accuracy
all emb	**0.94**	**0.936**	**0.915**	**0.932**	**0.924**	**0.911**
word and entity emd	0.924	0.911	0.896	0.925	0.906	0.889
word and context emd	0.898	0.890	0.872	0.889	0.876	0.848
word emd only	0.843	0.875	0.856	0.867	0.827	0.814

we see that *KABR* performs better than that without embeddings in terms of both the F1-measure and accuracy metrics. Specifically, for the Mozilla data set, *KABR* improves the best approach (i.e., *KABR with word and entity embeddings*) by 1.8%, 2.7% for F1-measure and accuracy, respectively. For the Eclipse data set, *KABR* improves the best approach (i.e., *KABR with word and entity embeddings*) by 2.1%, 1.8% for F1-measure and accuracy, respectively. For the Open Office data set, *KABR* improves the best approach (i.e., *KABR with word and entity embeddings*) by 1.9%, 1.3% for F1-measure and accuracy, respectively.

The results indicate that the combined embeddings generated by *KABR* works better than the three individual embedding. In addition, we see that *KABR* performs the best followed by *KABR* with entity and word embeddings, *KABR* with context and word embeddings, and *KABR* with word embedding only. So we can conclude that the integration of each embedding will introduce new effective information, and further improve the information retrieval ability.

4.4.2. RQ2: KABR for bug query with limited information

In our query reformulation approach, we first recognize text entities from the initial query (i.e., title and description), and then introduce knowledge from the bug-specific KG. The more linkable entities, the more knowledge can be obtained. However, in the actual bug analysis task, the query is generally a new bug report with limited information. In order to verify that *KABR* is also suitable in this situation, we compare on two test sets on the Mozilla project: $test_{IN}$ (i.e., title and description) and $test_{BT}$ (i.e., title only). As shown in Table 5, when the query is reduced with only the

Table 5
Comparison with limited information on Mozilla.

Approach	C4.5		DBR-CNN	
	F1	Accuracy	F1	Accuracy
initial query (Q_{IN})	0.827	0.845	0.846	0.875
observed behavior (Q_{OB})	0.872	0.878	0.891	0.899
OB and title (Q_{OB-BT})	0.890	0.883	0.907	0.911
title (Q_{BT})	0.823	0.829	0.851	0.877
KABR with title only	0.901	0.887	0.919	0.910
KABR	**0.921**	**0.917**	**0.94**	**0.936**

Table 6
Comparison with other query expansion approaches on Mozilla.

Approach	C4.5		DBR-CNN	
	F1	Accuracy	F1	Accuracy
initial query (Q_{IN})	0.827	0.845	0.846	0.875
synonym expansion (Q_{SE})	0.804	0.818	0.825	0.844
entity expansion (Q_{EE})	0.836	0.857	0.849	0.880
keyword expansion (Q_{KE})[a]	0.856	0.862	0.885	0.893
KABR	**0.921**	**0.917**	**0.94**	**0.936**

[a]We selected different amounts of Top-K (8–30) keywords for experiments. When K = 20, C4.5 based method achieves the best. When K = 15, DBR-CNN based method achieves the best.

title, the accuracy of *KABR* based on *C4.5* decreases by 0.3% and F1-measure decreases by 0.2%, the accuracy of *KABR* based on *DBR-CNN* decreases by 0.21% and F1-measure decreases by 0.26%, respectively, which are not large. However, whether based on *DBR-CNN* or *C4.5*, they are still much better than retrieval directly with the initial query.

We continue to compare *KABR* with three bug report reduction strategies (i.e., Q_{OB}, Q_{OB-BT} and Q_{BT}). When using *C4.5* as the classifier, the effect of Q_{BT} is slightly reduced. Both Q_{OB} and Q_{OB-BT} strategies indeed improve the retrieval and implement effective bug reformulation. This confirms that the title and observed behavior contain important information, which is consistent with Chaparro et al.'s work.

However, on the one hand *KABR* performs better than the above three strategies on both *C4.5* and *DBR-CNN* classifiers.

Compared with Q_{OB-BT}, which is the best among them, *KABR* improves accuracy by 3.9% and F1-measure by 3.5% for *C4.5*, improves accuracy by 2.7% and F1-measure by 3.6% for *DBR-CNN*. On the other hand, when only the title is provided, bug report reformulation with *KABR* outperforms without it. Therefore, we can conclude that *KABR* is useful for bug analysis on the query with limited information.

4.4.3. RQ3: KABR vs. other query expansion approaches

Different from the general query report reformulation approaches, *KABR* represent the new bug query in the form of embedding to keep the original sequence structure in the query.

The comparison results with other query reformulation approaches are shown in Table 6. We first compare *KABR* with the other three query expansion strategies (i.e., synonym expansion, entity expansion and keyword expansion). We can find that both the accuracy and F1-measure values are improved by the latter two strategies. In contrast, the accuracy and F1-measure values are decreased by synonym expansion. The opposite trend shows that the information obtained from the bug data set is more reliable than information external to the data set. We observe between entity expansion and keyword expansion. Both strategies select relevant terms (entities or keywords) from the same bug-specific KG data set and insert them directly into the initial query in the form of words. For a single query, the number of entities introduced far exceeds the number of keywords, but for both retrieval models, keyword expansion is better than entity expansion. At the same time, for keyword expansion, when the best effect is achieved, the experimental performance becomes worse with the further increase of K value. We argue that if the expanded information is integrated into the initial query in the state of meaningless word sequence, the effect of bug report reformulation is limited by the amount of information. Further compared *KABR* with keyword expansion based on *DBR-CNN*, *KABR* improves accuracy by 4.3%, and F1-measure by 5.5%. Compared *KABR* with keyword expansion based on *C4.5*, *KABR* improves accuracy by 5.5%, and F1-measure by 6.5%. So we can conclude that by leveraging multi-level embedding, *KABR* can better embed the external knowledge into the structural information of the query itself.

Through the duplicate bug detection task, different reformulation strategies are analyzed. To sum up, for *KABR*, both *KABR-C* and *KABR-S* models are optimal when integrating three-level embeddings and the attention mechanism. *KABR-S* is better than *KABR-C*. For *C4.5* and *DBR-CNN*, when query reduction is applied, that is, when the bug title and observed behavior are extracted from the initial bug report as a new query (a.k.a. Q_{OB-BT}), both *C4.5* and *DBR-CNN* models obtain optimal values. In the best case, we further perform statistical tests on the above three models to observe the advantages of *KABR*.

In terms of F1-measure, there is a significant difference between different approaches [F(2,36) = 12.25, $P < 0.001$, $\eta^2 P = 0.405$], and the post-hoc comparison results (P=0.185) show that there is no significant difference between *C4.5* and *DBR-CNN*, *C4.5* was significantly different from *KABR-S* ($P < 0.001$, Cohen's $d = -1.794$). Significant difference between *DBR-CNN* and *KABR-S* ($P = 0.015$, Cohen's $d = -1.241$) (see Table 7).

In terms of accuracy, the difference between different approaches is significant [F(2,36) = 7.758, $P < 0.002$, $\eta^2 P = 0.301$], post-hoc comparison results show: *C4.5* and *DBR-CNN* have no significant difference ($P = 0.470$), *C4.5* was significantly different from *KABR-S* ($P = 0.001$, Cohen's $d = -1.532$). The difference between *DBR-CNN* and *KABR-S* is marginally significant ($P = 0.058$, Cohen's $d = -0.879$). The results of statistical tests confirm that *KABR* is indeed significantly better than the state-of-the-art bug report reformulation approaches.

5. Threats to validity

In this section, we discuss possible threats to our study, including construct threats, external threats and internal threats.

- **Construct threats:** In our study, we evaluated *KABR* using two metrics (i.e., F1-measure and Accuracy), which are widely used in duplicate bug detection approaches. Other metrics may get different empirical results.
- **Internal threats:** One of the primary threats to the internal validity is the performance of underlying tools (i.e., NER and entity linking) . We use the *DBNER* approach for NER in the query. Although it has proved to be powerful, the *DBNER* approach may not provide accurate results for some cross-projects NER tasks because of a gap between projects due to different functions and environments. We introduce prior knowledge from the bug-specific KG through entity linking. Although we enrich the KG through various methods, there always be a few unlinkable entities in the query, which affects the subsequent information acquisition. We will further expand the scale of the bug-specific KG, and apply pre-training models with KGs to improve the accuracy of NER and entity linking.
- **External threats:** (1) We evaluated our approach on three popular open source projects (i.e., Eclipse, Mozilla and Open Office). A larger set of queries and more projects would strengthen the results from this perspective; (2) We only used the duplicate bug report detection task for study. In the future, we will investigate how the *KABR* help other IR-based bug analysis tasks.

6. Conclusions and future work

In this paper, we proposed a knowledge-aware bug report reformulation approach (a.k.a, *KABR*) for bug analysis. We first construct a bug-specific knowledge graph by extracting a large amount of bug knowledge from historical bug reports. Then, we leverage multi-level embeddings (i.e., word embedding from bug data, entity embedding and context embedding from the bug-specific KG) to generate a uniform bug report embedding to represent the bug query. The experiments on a typical bug analysis task, i.e., duplicate bug report detection, show that *KABR* is effective to introduce relevant information for bug report reformulation.

In the future, we will utilize more bug data from different projects to enrich the bug-specific KG. In this way, the entity embedding and context embedding can be more accurate. In addition, we will perform more studies on different bug analysis tasks to further show the effectiveness of *KABR*.

Table 7
Statistical tests of different reformulation strategies on Mozilla.

Approach	Mean Difference		SE		*t*		Cohen's *d*		*p*	
	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy	F1	Accuracy
C4.5 (Q_{OB-BT}) vs. DBR-CNN (Q_{OB-BT})	−0.025	−0.017	0.013	0.012	−1.93	−1.446	−0.774	−0.631	0.185	0.47
C4.5 (Q_{OB-BT}) vs. KABR-S	−0.063	−0.045	0.013	0.012	−4.912	−3.896	−1.794	−1.532	<0.001	0.001
DBR-CNN (Q_{OB-BT}) vs. KABR-S	−0.038	−0.029	0.013	0.012	−2.982	−2.45	−1.241	−0.879	0.015	0.058

CRediT authorship contribution statement

Cheng Zhou: Methodology, Software, Validation, Writing – original draft, Writing – review & editing. **Bin Li:** Conceptualization, Methodology, Supervision, Project administration. **Xiaobing Sun:** Methodology, Supervision, Writing – review & editing. **Sheng Yu:** Validation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgments

This work is supported by the National Natural Science Foundation of China (No. 61872312, No. 61972335, No. 62002309); the Yangzhou city-Yangzhou University Science and Technology Cooperation Fund Project (No. YZU201803, No. YZU201902); the Six Talent Peaks Project in Jiangsu Province (No. RJFW-053), the Jiangsu "333" Project; the Natural Science Foundation of the Jiangsu Higher Education Institutions of China (No. 20KJB520016); the Open Funds of State Key Laboratory for Novel Software Technology of Nanjing University (No. KFKT2020B15, No. KFKT2020B16) and Yangzhou University Top-level Talents Support Program (2019).

References

Aggarwal, Karan, Timbers, Finbarr, Rutgers, Tanner, Hindle, Abram, Stroulia, Eleni, Greiner, Russell, 2017. Detecting duplicate bug reports with software engineering domain knowledge. J. Softw.: Evol. Process 29 (3).

Bacchelli, Alberto, Bird, Christian, 2013. Expectations, outcomes, and challenges of modern code review. In: 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013. pp. 712–721.

Bodden, Eric, Schäfer, Wilhelm, van Deursen, Arie, Zisman, Andrea (Eds.), 2017. Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017. ACM.

Bordes, Antoine, Usunier, Nicolas, García-Durán, Alberto, Weston, Jason, Yakhnenko, Oksana, 2013. Translating embeddings for modeling multi-relational data. In: Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a Meeting Held December 5-8, 2013, Lake Tahoe, Nevada, United States. pp. 2787–2795.

Budhiraja, Amar, Dutta, Kartik, Reddy, Raghu, Shrivastava, Manish, 2018. DWEN: deep word embedding network for duplicate bug report detection in software repositories. In: Chaudron, Michel, Crnkovic, Ivica, Chechik, Marsha, Harman, Mark (Eds.), Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018. ACM, pp. 193–194.

Chaparro, Oscar, Florez, Juan Manuel, Marcus, Andrian, 2016. On the vocabulary agreement in software issue descriptions. In: 2016 IEEE International Conference on Software Maintenance and Evolution, ICSME 2016, Raleigh, NC, USA, October 2-7, 2016. IEEE Computer Society, pp. 448–452.

Chaparro, Oscar, Florez, Juan Manuel, Marcus, Andrian, 2019a. Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empir. Softw. Eng. 24 (5), 2947–3007.

Chaparro, Oscar, Florez, Juan Manuel, Singh, Unnati, Marcus, Andrian, 2019b. Reformulating queries for duplicate bug report detection. In: 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019. pp. 218–229.

Chen, Dingshan, Li, Bin, Zhou, Cheng, Zhu, Xuanrui, 2019. Automatically identifying bug entities and relations for bug analysis. In: 2019 IEEE 1st International Workshop on Intelligent Bug Fixing. IBF, IEEE, pp. 39–43.

Chen, Samuel, Xie, Shengyi, Chen, Qingqiang, 2020. Integrated embedding approach for knowledge base completion with CNN. Inf. Technol. Control. 49 (4), 622–642.

Church, Kenneth Ward, Hanks, Patrick, 1989. Word association norms, mutual information and lexicography. In: 27th Annual Meeting of the Association for Computational Linguistics, 26-29 June 1989, University of British Columbia, Vancouver, BC, Canada, Proceedings. pp. 76–83.

Dalton, Jeffrey, Dietz, Laura, Allan, James, 2014. Entity query feature expansion using knowledge base links. In: The 37th International ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '14, Gold Coast , QLD, Australia - July 06 - 11, 2014. pp. 365–374.

Damani, Om P., 2013. Improving pointwise mutual information (PMI) by incorporating significant co-occurrence. In: Proceedings of the Seventeenth Conference on Computational Natural Language Learning, CoNLL 2013, Sofia, Bulgaria, August 8-9, 2013. pp. 20–28.

Davies, Steven, Roper, Marc, 2014. What's in a bug report? In: 2014 ACM-IEEE International Symposium on Empirical Software Engineering and Measurement, ESEM '14, Torino, Italy, September 18-19, 2014. pp. 26:1–26:10.

Ebrahimi, Neda, Trabelsi, Abdelaziz, Islam, Md. Shariful, Hamou-Lhadj, Abdelwahab, Khanmohammadi, Kobra, 2019. An HMM-based approach for automatic detection and classification of duplicate bug reports. Inf. Softw. Technol. 113, 98–109.

Elmishali, Amir, Stern, Roni, Kalech, Meir, 2019. DeBGUer: A tool for bug prediction and diagnosis. In: The Thirty-Third AAAI Conference on Artificial Intelligence, AAAI 2019, the Thirty-First Innovative Applications of Artificial Intelligence Conference, IAAI 2019, the Ninth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2019, Honolulu, Hawaii, USA, January 27 - February 1, 2019. pp. 9446–9451.

Fensel, Dieter, Simsek, Umutcan, Angele, Kevin, Huaman, Elwin, Kärle, Elias, Panasiuk, Oleksandra, Toma, Ioan, Umbrich, Jürgen, Wahler, Alexander, 2020. Knowledge Graphs - Methodology, Tools and Selected Use Cases. Springer.

Gao, Jie, Xin, Xin, Liu, Junshuai, Wang, Rui, Lu, Jing, Li, Biao, Fan, Xin, Guo, Ping, 2018. Fine-grained deep knowledge-aware network for news recommendation with self-attention. In: 2018 IEEE/WIC/ACM International Conference on Web Intelligence, WI 2018, Santiago, Chile, December 3-6, 2018. pp. 81–88.

Haiduc, Sonia, Bavota, Gabriele, Marcus, Andrian, Oliveto, Rocco, Lucia, Andrea De, Menzies, Tim, 2013. Automatic query reformulations for text retrieval in software engineering. In: Notkin, David, Cheng, Betty H.C., Pohl, Klaus (Eds.), 35th International Conference on Software Engineering, ICSE '13, San Francisco, CA, USA, May 18-26, 2013. IEEE Computer Society, pp. 842–851.

He, Jianjun, Xu, Ling, Yan, Meng, Xia, Xin, Lei, Yan, 2020. Duplicate bug report detection using dual-channel convolutional neural networks. In: ICPC '20: 28th International Conference on Program Comprehension, Seoul, Republic of Korea, July 13-15, 2020. ACM, pp. 117–127.

Hindle, Abram, Alipour, Anahita, Stroulia, Eleni, 2016. A contextual approach towards more accurate duplicate bug report detection and ranking. Empir. Softw. Eng. 21 (2), 368–410.

Hou, Wei, Tao, Xian, Xu, De, 2021. Combining prior knowledge with CNN for weak scratch inspection of optical components. IEEE Trans. Instrum. Meas. 70, 1–11.

Hu, Dongyang, Chen, Ming, Wang, Tao, Chang, Junsheng, Yin, Gang, Yu, Yue, Zhang, Yang, 2018. Recommending similar bug reports: A novel approach using document embedding model. In: 25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018. pp. 725–726.

Jiang, Lin, Liu, Hui, Jiang, He, Zhang, Lu, Mei, Hong, 2020. Heuristic and neural network based prediction of project-specific API member access. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2020.3017794.

Kim, Yoon, 2014. Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing, EMNLP 2014, October 25-29, 2014, Doha, Qatar, a Meeting of SIGDAT, a Special Interest Group of the ACL. pp. 1746–1751.

Kim, Sunghun, Jr., E. James Whitehead, 2006. How long did it take to fix bugs? In: Proceedings of the 2006 International Workshop on Mining Software Repositories, MSR 2006, Shanghai, China, May 22-23, 2006. pp. 173–174.

Kim, Misoo, Lee, Eunseok, 2019. A novel approach to automatic query reformulation for IR-based bug localization. In: Hung, Chih-Cheng, Papadopoulos, George A. (Eds.), Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing, SAC 2019, Limassol, Cyprus, April 8-12, 2019. ACM, pp. 1752–1759.

Kim, Misoo, Lee, Eunseok, 2020. Manq: Many-objective optimization-based automatic query reduction for IR-based bug localization. Inf. Softw. Technol. 125, 106334.

Kingma, Diederik P., Ba, Jimmy, 2015. Adam: A method for stochastic optimization. In: 3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings.

Lazar, Alina, Ritchey, Sarah, Sharif, Bonita, 2014. Improving the accuracy of duplicate bug report detection using textual similarity measures. In: Devanbu, Premkumar T., Kim, Sung, Pinzger, Martin (Eds.), 11th Working Conference on Mining Software Repositories, MSR 2014, Proceedings, May 31 - June 1, 2014, Hyderabad, India. ACM, pp. 308–311.

Lee, Sun-Ro, Heo, Min-Jae, Lee, Chan-Gun, Kim, Milhan, Jeong, Gaeul, 2017. Applying deep learning based automatic bug triager to industrial projects. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering, ESEC/FSE 2017, Paderborn, Germany, September 4-8, 2017. pp. 926–931.

Li, Mingchen, Clinton, Gabtone., Miao, Yijia, Gao, Feng, 2020. Short text classification via knowledge powered attention with similarity matrix based CNN. CoRR arXiv:2002.03350.

Li, Hongwei, Li, Sirui, Sun, Jiamou, Xing, Zhenchang, Peng, Xin, Liu, Mingwei, Zhao, Xuejiao, 2018. Improving API caveats accessibility by mining API caveats knowledge graph. In: 2018 IEEE International Conference on Software Maintenance and Evolution, ICSME 2018, Madrid, Spain, September 23-29, 2018. pp. 183–193.

Lin, Zeqi, Xie, Bing, Zou, Yanzhen, Zhao, Junfeng, Li, Xuan-Dong, Wei, Jun, Sun, Hailong, Yin, Gang, 2017. Intelligent development environment and software knowledge graph. J. Comput. Sci. Tech. 32 (2), 242–249.

Lin, Meng-Jie, Yang, Cheng-Zen, Lee, Chao-Yuan, Chen, Chun-Chang, 2016. Enhancements for duplication detection in bug reports with manifold correlation features. J. Syst. Softw. 121, 223–233.

Liu, Xitong, Fang, Hui, 2015. Latent entity space: a novel retrieval approach for entity-bearing queries. Inf. Retr. J. 18 (6), 473–503.

Liu, Zhenyu, Huang, Haiwei, Lu, Chaohong, Lyu, Shengfei, 2020. Multichannel CNN with attention for text classification. CoRR arXiv:2006.16174.

Liu, Hui, Jin, Jiahao, Xu, Zhifeng, Bu, Yifang, Zou, Yanzhen, Zhang, Lu, 2019. Deep learning based code smell detection. IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2019.2936376.

Liu, Jason, Kim, Seohyun, Murali, Vijayaraghavan, Chaudhuri, Swarat, Chandra, Satish, 2019a. Neural query expansion for code search. In: Mattson, Tim, Muzahid, Abdullah, Solar-Lezama, Armando (Eds.), Proceedings of the 3rd ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, MAPL@PLDI 2019, Phoenix, AZ, USA, June 22, 2019. ACM, pp. 29–37.

Liu, Guangliang, Lu, Yang, Shi, Ke, Chang, Jingfei, Wei, Xing, 2019b. Convolutional neural networks-based locating relevant buggy code files for bug reports affected by data imbalance. IEEE Access 7, 131304–131316.

Liu, Hui, Shen, Minzhu, Zhu, Jiaqi, Niu, Nan, Li, Ge, Zhang, Lu, 2020. Deep learning based program generation from requirements text: Are we there yet? IEEE Trans. Softw. Eng. 1. http://dx.doi.org/10.1109/TSE.2020.3018481.

Liu, Kaiping, Tan, Hee Beng Kuan, Zhang, Hongyu, 2013. Has this bug been reported? In: Lämmel, Ralf, Oliveto, Rocco, Robbes, Romain (Eds.), 20th Working Conference on Reverse Engineering, WCRE 2013, Koblenz, Germany, October 14-17, 2013. IEEE Computer Society, pp. 82–91.

Lu, X. Allan, Keefer, Robert B., 1994. Query expansion/reduction and its impact on retrieval effectiveness. In: Harman, Donna K. (Ed.), Proceedings of the Third Text REtrieval Conference, TREC 1994, Gaithersburg, Maryland, USA, November 2-4, 1994. In: NIST Special Publication, 500–225, National Institute of Standards and Technology (NIST), pp. 231–240.

Lu, Meili, Sun, Xiaobing, Wang, Shaowei, Lo, David, Duan, Yucong, 2015. Query expansion via WordNet for effective code search. In: Guéhéneuc, Yann-Gaël, Adams, Bram, Serebrenik, Alexander (Eds.), 22nd IEEE International Conference on Software Analysis, Evolution, and Reengineering, SANER 2015, Montreal, QC, Canada, March 2-6, 2015. IEEE Computer Society, pp. 545–549.

McCallum, Andrew, Li, Wei, 2003. Early results for named entity recognition with conditional random fields, feature induction and web-enhanced lexicons. In: Proceedings of the Seventh Conference on Natural Language Learning, CoNLL 2003, Held in Cooperation with HLT-NAACL 2003, Edmonton, Canada, May 31 - June 1, 2003. pp. 188–191.

Nayrolles, Mathieu, Hamou-Lhadj, Abdelwahab, 2018. Towards a classification of bugs to facilitate software maintainability tasks. In: Proceedings of the 1st International Workshop on Software Qualities and their Dependencies, SQUADE@ICSE 2018, Gothenburg, Sweden, May 28, 2018. pp. 25–32.

Neysiani, Behzad Soleimani, Babamir, Seyed Morteza, Aritsugi, Masayoshi, 2020. Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems. Inf. Softw. Technol. 126, 106344.

Nguyen, Anh Tuan, Nguyen, Tung Thanh, Nguyen, Tien N., Lo, David, Sun, Chengnian, 2012. Duplicate bug report detection with a combination of information retrieval and topic modeling. In: IEEE/ACM International Conference on Automated Software Engineering, ASE'12, Essen, Germany, September 3-7, 2012. pp. 70–79.

Nie, Liming, Jiang, He, Ren, Zhilei, Sun, Zeyi, Li, Xiaochen, 2017. Query expansion based on crowd knowledge for code search. CoRR arXiv:1703.01443.

Rahman, Mohammad Masudur, Roy, Chanchal K., 2017. STRICT: information retrieval based search term identification for concept location. In: IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER 2017, Klagenfurt, Austria, February 20-24, 2017. pp. 79–90.

Rahman, Mohammad Masudur, Roy, Chanchal K., 2018a. Improving IR-based bug localization with context-aware query reformulation. In: Leavens, Gary T., Garcia, Alessandro, Pasareanu, Corina S. (Eds.), Proceedings of the 2018 ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, ESEC/SIGSOFT FSE 2018, Lake Buena Vista, FL, USA, November 04-09, 2018. ACM, pp. 621–632.

Rahman, Mohammad Masudur, Roy, Chanchal K., 2018b. QUICKAR: automatic query reformulation for concept location using crowdsourced knowledge. CoRR arXiv:1807.02964.

Rahman, Mohammad Masudur, Roy, Chanchal K., Lo, David, 2019. Automatic query reformulation for code search using crowdsourced knowledge. Empir. Softw. Eng. 24 (4), 1869–1924.

Rakha, Mohamed Sami, Bezemer, Cor-Paul, Hassan, Ahmed E., 2018. Revisiting the performance evaluation of automated approaches for the retrieval of duplicate issue reports. IEEE Trans. Softw. Eng. 44 (12), 1245–1268.

Rastkar, Sarah, Murphy, Gail C., Murray, Gabriel, 2014. Automatic summarization of bug reports. IEEE Trans. Softw. Eng. 40 (4), 366–380.

Sabou, Marta, Ekaputra, Fajar J., Ionescu, Tudor B., Musil, Juergen, Schall, Daniel, Haller, Kevin, Friedl, Armin, Biffl, Stefan, 2018. Exploring enterprise knowledge graphs: A use case in software engineering. In: The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings. pp. 560–575.

Shen, Wei, Wang, Jianyong, Han, Jiawei, 2015. Entity linking with a knowledge base: Issues, techniques, and solutions. IEEE Trans. Knowl. Data Eng. 27 (2), 443–460.

Sisman, Bunyamin, Kak, Avinash C., 2013. Assisting code search with automatic query reformulation for bug localization. In: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR '13, San Francisco, CA, USA, May 18-19, 2013. pp. 309–318.

Thung, Ferdian, Lo, David, Jiang, Lingxiao, 2012. Automatic defect categorization. In: 19th Working Conference on Reverse Engineering, WCRE 2012, Kingston, on, Canada, October 15-18, 2012. pp. 205–214.

Umer, Qasim, Liu, Hui, Illahi, Inam, 2020. CNN-based automatic prioritization of bug reports. IEEE Trans. Reliab. 69 (4), 1341–1354.

Vaswani, Ashish, Shazeer, Noam, Parmar, Niki, Uszkoreit, Jakob, Jones, Llion, Gomez, Aidan N., Kaiser, Lukasz, Polosukhin, Illia, 2017. Attention is all you need. In: Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, 4-9 December 2017, Long Beach, CA, USA. pp. 5998–6008.

Wang, Lu, Sun, Xiaobing, Wang, Jingwei, Duan, Yucong, Li, Bin, 2017a. Construct bug knowledge graph for bug resolution: poster. In: Proceedings of the 39th International Conference on Software Engineering, ICSE 2017, Buenos Aires, Argentina, May 20-28, 2017 - Companion Volume. pp. 189–191.

Wang, Jin, Wang, Zhongyuan, Zhang, Dawei, Yan, Jun, 2017b. Combining knowledge with deep convolutional neural networks for short text classification. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 2915–2921.

Wang, Xin, Wu, Hao, Hsu, Ching-Hsien, 2019. Mashup-oriented API recommendation via random walk on knowledge graph. IEEE Access 7, 7651–7662.

Wang, Hongwei, Zhang, Fuzheng, Xie, Xing, Guo, Minyi, 2018. DKN: deep knowledge-aware network for news recommendation. In: Proceedings of the 2018 World Wide Web Conference on World Wide Web, WWW 2018, Lyon, France, April 23-27, 2018. pp. 1835–1844.

Xiao, Guanping, Du, Xiaoting, Sui, Yulei, Yue, Tao, 2020. HINDBR: heterogeneous information network based duplicate bug report prediction. In: Vieira, Marco, Madeira, Henrique, Antunes, Nuno, Zheng, Zheng (Eds.), 31st IEEE International Symposium on Software Reliability Engineering, ISSRE 2020, Coimbra, Portugal, October 12-15, 2020. IEEE, pp. 195–206.

Xiao, Yan, Keung, Jacky, Bennin, Kwabena Ebo, Mi, Qing, 2019. Improving bug localization with word embedding and enhanced convolutional neural networks. Inf. Softw. Technol. 105, 17–29.

Xie, Qi, Wen, Zhiyuan, Zhu, Jieming, Gao, Cuiyun, Zheng, Zibin, 2018. Detecting duplicate bug reports with convolutional neural networks. In: 25th Asia-Pacific Software Engineering Conference, APSEC 2018, Nara, Japan, December 4-7, 2018. pp. 416–425.

Xu, Jiacheng, Qiu, Xipeng, Chen, Kan, Huang, Xuanjing, 2017. Knowledge graph representation with jointly structural and textual encoding. In: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI 2017, Melbourne, Australia, August 19-25, 2017. pp. 1318–1324.

Yang, Xinli, Lo, David, Xia, Xin, Bao, Lingfeng, Sun, Jianling, 2016. Combining word embedding with information retrieval to recommend similar bug reports. In: 27th IEEE International Symposium on Software Reliability Engineering, ISSRE 2016, Ottawa, on, Canada, October 23-27, 2016. pp. 127–137.

Yang, Jinqiu, Tan, Lin, 2014. SWordNet: Inferring semantically related words from software context. Empir. Softw. Eng. 19 (6), 1856–1886.

Ye, Xin, Bunescu, Razvan C., Liu, Chang, 2014. Learning to rank relevant files for bug reports using domain knowledge. In: Proceedings of the 22nd ACM SIGSOFT International Symposium on Foundations of Software Engineering, (FSE-22), Hong Kong, China, November 16 - 22, 2014. pp. 689–699.

Ye, Xin, Fang, Fan, Wu, John, Bunescu, Razvan C., Liu, Chang, 2018. Bug report classification using LSTM architecture for more accurate software defect locating. In: 17th IEEE International Conference on Machine Learning and Applications, ICMLA 2018, Orlando, FL, USA, December 17-20, 2018. pp. 1438–1445.

Zaidi, Syed Farhan Alam, Awan, Faraz Malik, Lee, Minsoo, Woo, Honguk, Lee, Chan-Gun, 2020. Applying convolutional neural networks with different word representation techniques to recommend bug fixers. IEEE Access 8, 213729–213747.

Zhang, Quanshi, Cao, Ruiming, Shi, Feng, Wu, Ying Nian, Zhu, Song-Chun, 2018. Interpreting CNN knowledge via an explanatory graph. In: McIlraith, Sheila A., Weinberger, Kilian Q. (Eds.), Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence, (AAAI-18), the 30th Innovative Applications of Artificial Intelligence (IAAI-18), and the 8th AAAI Symposium on Educational Advances in Artificial Intelligence (EAAI-18), New Orleans, Louisiana, USA, February 2-7, 2018. AAAI Press, pp. 4454–4463.

Zhang, Zhengyan, Han, Xu, Liu, Zhiyuan, Jiang, Xin, Sun, Maosong, Liu, Qun, 2019a. ERNIE: enhanced language representation with informative entities. In: Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers. pp. 1441–1451.

Zhang, Wen, Li, Ziqiang, Wang, Qing, Li, Juan, 2019b. FineLocator: A novel approach to method-level fine-grained bug localization by query expansion. Inf. Softw. Technol. 110, 121–135.

Zhao, Yuan, He, Tieke, Chen, Zhenyu, 2019a. A unified framework for bug report assignment. Int. J. Softw. Eng. Knowl. Eng. 29 (4), 607–628.

Zhao, Yanjie, Wang, Haoyu, Ma, Lei, Liu, Yuxin, Li, Li, Grundy, John, 2019b. Knowledge graphing git repositories: A preliminary study. In: 26th IEEE International Conference on Software Analysis, Evolution and Reengineering, SANER 2019, Hangzhou, China, February 24-27, 2019. pp. 599–603.

Zhou, Cheng, Li, Bin, Sun, Xiaobing, 2020. Improving software bug-specific named entity recognition with deep neural network. J. Syst. Softw. 165, 110572.

Zhou, Peng, Shi, Wei, Tian, Jun, Qi, Zhenyu, Li, Bingchen, Hao, Hongwei, Xu, Bo, 2016. Attention-based bidirectional long short-term memory networks for relation classification. In: Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics, ACL 2016, August 7-12, 2016, Berlin, Germany, Volume 2: Short Papers.

Cheng Zhou is a Ph.D. student in School of Information Engineering, Yangzhou University in China. Her current research interests include intelligent bug fixing and software data analysis. (Email: mailto:canorcheng@foxmail.com)

Bin Li is a professor in School of Information Engineering, Yangzhou University in China. His current research interests include software engineering, artificial intelligence, etc. (Email: mailto:lb@yzu.edu.cn)

Xiaobing Sun is a professor in School of Information Engineering, Yangzhou University in China. His current research interests include intelligent software engineering, software data analytics. (Email: mailto:xbsun@yzu.edu.cn)

Sheng Yu is a master student in School of Information Engineering, Yangzhou University in China. His current research interests are mainly the research and application of knowledge extraction for software bugs. (Email: mailto:2822863494@qq.com)