KeyTitle: Towards Better Bug Report Title Generation by Keywords Planning

Qianshuang Meng[1], Weiqin Zou[1,2*], Biyu Cai[1], Jingxuan Zhang[1]

[1]Department of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, China.
[2]State Key Lab. for Novel Software Technology, Nanjing University, China.

*Corresponding author(s). E-mail(s): weiqin@nuaa.edu.cn;
Contributing authors: qs_meng@nuaa.edu.cn; caibiyu@nuaa.edu.cn; jxzhang@nuaa.edu.cn;

Abstract

Bug reports play an important role in the software development and maintenance process. As the eye of a bug report, a concise and fluent title is always preferred and expected by developers as it could help them quickly seize the problem point and make better decisions in handling the bugs. However, in practice, not all titles filled by bug reporters are found to be of high quality; some may not carry essential bug-related information, and some may be hard to understand or contain extra noise. With the aim to reduce the burden of bug reporters and ease developers' life in handling bugs, we propose a deep learning-based technique named KeyTitle, to automatically generate a title for a given bug report. KeyTitle formulates the title generation problem as a one-sentence summarization task. It could be viewed as a Seq2Seq generation model (which generally directly generates target text based on source text) that incorporates keywords planning. Specifically, within KeyTitle, a transformer-based encoder-decoder model is enforced to generate a chain of keywords first from the detailed textual problem description, and then generate the target title by considering both these keywords and description content. Experiments over three large bug datasets collected from GitHub, Eclipse, and Apache shows that KeyTitle could outperform state-of-art title generation models relatively by up to 8.9-18.2%, 11.4-30.4%, and 13.0-18.0% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores; the titles generated by KeyTitle are also found to be better in terms of Relevance, Accuracy, Conciseness, Fluency in human evaluation. Besides generating titles from textual descriptions, KeyTitle is also found to have great potential in generating titles based on just a few keywords, a task that also has much value in bug reporting/handling practice.

Keywords: title generation, keywords planning, bug report, software engineering

1 Introduction

Bug reports play a key role in the software development and maintenance process. In general, a bug report typically includes information such as observed behavior, expected behavior, a description of a problem, steps to reproduce the issue, and a title of the bug report, etc [1, 2]. With the help of bug reports, developers can identify software defects [3], prioritize bug fixes [4], track the status of bugs [5], and timely resolve bugs to improve the quality of software products.

Title, as a high-level summary of a bug report, is the part that developers would see at first glance. It gives bug fixers a direct idea of what the bug report is about and may also largely affect whether potential contributors would decide to work on the bug in software projects adopting open-sourced development approach. Hence, drafting a good title that summarizes a bug report precisely and concisely is quite important, for both bug reporters/users who hope their bugs got fixed timely and developers who need to manage a number of bug reports.

In current practice, the title of a bug report is mainly written by end-users or product developers/testers. Despite there exist some guidelines like bug report templates[1] provided for users to follow in reporting bugs, the titles of existing bug reports are still found to be of varied quality [6]. Some reasons for low-quality titles include the inherent difficulty in summarization (which often requires people to spare more time rethinking the problem and reorganizing the key points), the lack of experience of bug reporters, and the overlook of title importance [6].

To help reduce the burden of bug reporters and ease the life of developers in handling bugs, some researchers try to automatically generate a title for a bug report based on the detailed description of the bug [6–8]. The general idea of these studies is to treat title generation as a Seq2Seq (sequence to sequence) generation task [9], where the description and the title is usually treated as the input and the output for the Seq2Seq model separately.

One problem with current title generation research is that the adopted end-to-end generation process often takes the whole description as plain text to generate a title without considering its information structure and filtering possible noise within it. As an essential part of a bug report, a description usually contains different components, like plain text, code, and stack trace. The meaning of a word appearing in a code segment may differ much from that in a normal text (e.g., "while" generally represents "a loop structure" in code but "in spite of the fact that" in normal text). Additionally, the embedded stack trace tends to contain much redundant and repeated information (e.g., "java.lang.xxx"). Without taking the above points into consideration would adversely affect the performance of title generation models.

Another problem with this kind of technology is that it would suffer from the Out-Of-Vocabulary (OOV) problem which is common for software development where

[1] https://bugs.eclipse.org/bugs/page.cgi?id=bug-writing.html

new words always appear. Despite Chen et al. [6] try to directly copy tokens tagged by human from input to resolve the OOV problem, the limited human tags (only for version numbers and identifiers) is not likely to well adapt to future software development (we still witness the OOV problem using iTAPE proposed in [6], more details in Section 5.1).

To address the above problems, we propose KeyTitle, a transformer-based framework to generate titles from descriptions with keywords planning, where keywords planning is used to help solve the first problem and the transformer in which words are cut into subwords could help solve the OOV problem [10, 11]. More specifically, we first extract a chain of keywords that reflect the main topic of a bug from the description. Based on the historical <description, keywords, title> data, we force the transformer model to generate a chain of keywords first, then to generate the title guided by the keywords (i.e., keywords planning). Compared to general seq2seq model that directly generates a title from a description, this could make our model learn more from both the descriptions and keywords because of the autoregressive mechanism [12], and hence generate titles with higher quality.

To fully verify the performance of KeyTitle, we use three large bug datasets collected from different platforms that adopt different bug tracking systems to manage their bugs, i.e., project issues from GitHub, Eclipse bug reports from Bugzilla[2], and Apache bug reports from Jira[3]. The experiment results show that KeyTitle could outperform the state-of-the-art title generation approaches relatively by up to 8.9-18.2%, 11.4-30.4%, and 13.0-18.0% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Meanwhile, the titles generated by KeyTitle get higher scores in terms of Relevance, Accuracy, Conciseness, Fluency in manual evaluation. Besides, KeyTitle is also found to have great potential in generating titles directly from just a few keywords. Our main contributions are as follows:

- We propose a transformer-based framework KeyTitle to generate a title for a bug report by keywords planning.
- Experiments on three large bug datasets shows that KeyTitle could outperform state-of-the-art baselines in typical evaluation metrics and also achieve better performance in human evaluation.
- Besides generating titles from descriptions, we also make the first attempt to explore the possibility of KeyTitle to generate a title directly by giving few keywords.

The rest of the paper is organized as follows: Section 2 introduces the background. Section 3 details the framework of KeyTitle. We describe our experimental setup and results separately in Section 4 and 5. Section 6 further discusses the use of GPTs and the threats to validity. We introduce the related work in and conclude our study in Section 7 and 8.

[2]https://www.bugzilla.org
[3]https://jira.atlassian.com

Table 1: Title Examples with Low Quality

Example 1	**Example 2**	**Example 3**
Title: web.xml issue **Description:** i could not load web.xml file ,deployment descriptor is not running i have done all cleaning, In Maven project or web project is not running please provide me necessary details and solution to resolve it.	**Title:** Missing code suggestions for result of a method within the parameter list of another method when annotation processing is enabled (no annotation processor neccessary) **Description:** Created attachment 278058 Screenshot without code suggestions Steps to reproduce: 1. git clone https://github.com/Adrodoc/eclipse-bug-autocompletion.git 2. Import eclipse...	**Title:** Support after(), around(), cflow() for handler() pointcut **Description:** When trying to answer SO question, I noticed that the handler() pointcut only supports before() advice. In order to solve the given problem I would need something like cflow(handler()), though. I can write and compile that, but the corresponding advice never triggers...

2 Background

In this section, we first elaborate the quality problem of manually filled bug report titles by using three representative examples. Then we introduce the usage scenario of KeyTitle.

2.1 Low-quality Titles

Generally speaking, as a summary of a bug report, the title should accurately capture the key information of the bug, with which readers could get to the point in a few words. However, in practice, due to various reasons like lacking experience or overlooking the importance of good titles, it is common to see bug report titles failed to cover bug-related important information [6]. Based on our observation, we find that among those titles with low quality, some may be too short to carry crucial information; some may contain much redundancy and fail to convey the true concern; and some may lack naturalness and fluency. Below are three representative corresponding examples (in table 1).

As shown in Table 1, **Example 1** has a very short title, with only two words. It barely contains much useful information, except for indicating that it is about "web.xml". From the tile, readers cannot find out what the "web.xml issue" is really about. A good title should summarize the bug report accurately. In this example, a good title maybe should mention "fail to load", "deployment descriptor" or "Maven" according to the description. In general, it is unlikely for short titles like Example 1

to convey enough information and make readers understand the major concern of the bug.

The title of **Example 2** in Table 1 is too long for readers to digest. The really useful information is drowned in the long sentence. Readers may find it difficult to know the real intention and request. Maybe removing the "when" clause and the parentheses would make the sentence easier to understand. In this case, long titles may contain redundant information and not easy to understand.

The problem in **Example 3** in Table 1 is more subtle. It contains too many code related words in the title. Readers will get confused by so many specific functions of code. This may be because contributors themselves overlooked the importance of titles, and were only concerned with showing the issue encountered, forgetting to provide more general information about the issue.

The above-mentioned quality problems of human-filled titles inspire us to develop automatic title generation techniques, with the hope to assist reporters/developers in summarizing bug reports, understanding and fixing bugs, etc.

2.2 Usage Scenario

The goal of KeyTitle is to automatically generate titles with good quality as much as possible. Within KeyTitle, we enforce the model to generate keywords from the description first, then use the keywords to guide the generation of titles. In other words, KeyTitle actually supports two kinds of usage scenario, one is to generate titles from description (noted as description2title), the other is to generate titles directly based on some keywords (noted as keywords2title) that could be provided from the outside world rather than from the description, e.g., by users.

In the description2title scenario, when a user finishes writing the description of a bug report, KeyTitle can generate a fairly good title, that contains key information of the description and the text itself is natural-sounding and fluent (achieved by using the pre-trained language model that contains much general knowledge). Without worrying about the quality of generated title, the contributor could adopt the title with ease and focus more on providing other useful information related to the bug in the description.

In the keyword2title scenario, when a user wants to write a title but with only keywords in his/her mind. Without worrying about the organization of content and grammar issues, he/she just needs to input the keywords, and our model would try to infer their relations/connections, recover possible events, and then return a title that is expected by users. When retrieving related content or using a search engine, the title is usually the search object. However, not every user wants to input the whole sentence for searching, sometimes they just input a few words. How to bridge the gap between words and titles become a question now. If the intent of user is not accurately restored, the search effectiveness will be greatly reduced. In that case, KeyTitle, with the ability to generate titles from keywords, serves as a solution.

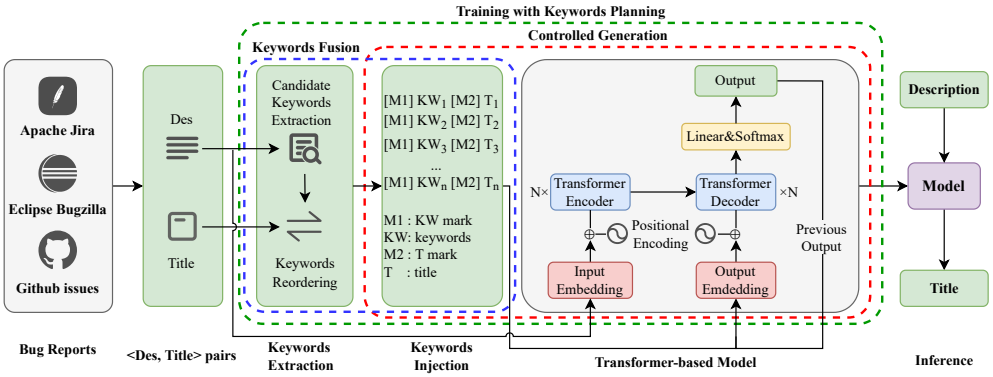

Fig. 1: Overview of KeyTitle

3 The Framework of KeyTitle

3.1 Overview

Following [6, 7], we also formulate title generation as a one-sentence summarization task, which also follows the convention that the title is typically submitted as a one-sentence summary in various bug tracking systems. Unlike existing studies that use Seq2Seq models to directly generate the title from the description of a bug report, we introduce a new training objective into the generation model to better generate titles, that is keywords planning. With keywords planning, the generation process would have two steps. The first step is to generate a chain of keywords that capture the main points of the description. Then, these keywords are used to guide the model to generate the title. We introduce keywords planning mainly by considering that existing studies generally ignore the information structure of the description (taking code as plain text) and fail to remove embedded noise, which adversely affect the quality of generated titles. Figure 1 shows the overview of KeyTitle. In general, KeyTitle can be viewed as a transformer-based encoder-decoder model enhanced with keywords planning. The reasons why we choose transformer rather than for example Bi-LSTM like iTAPE are that the transformer model could better solve the OOV problem and show great potential in many natural language processing tasks.

As shown in the figure, KeyTitle is first built upon e.g., Apache/Eclipse/GitHub bug reports. After obtaining the generation model, users only need to feed the model a new description, and the model would automatically generate a summary, i.e., a title, for them in the inference part. The whole model training process consists of two parts, namely Keywords Fusion and Controlled Generation. Keywords Fusion aims to inject the semantics of keywords into the titles, where the injected titles would work as the training target during model building. It includes two parts, i.e., Keywords Extraction and Keywords Injection. For the Controlled Generation part, we mainly build a standard transformer generation model based on the injected titles, where the model can be forced to generate keywords first, then the title. The details are as follows.

3.2 Keywords Fusion

In this part, we mainly do two things, one is called keywords extraction, i.e., extracting keywords from the descriptions; the other one is called keywords injection, i.e., injecting the extracted keywords into titles.

Keywords Extraction. To alleviate the problems of ignoring the information structure and lacking noise removal of the description, we propose a keyword extraction method to make use of keywords that represent the key points of the description to guide the generation model training. As the quality of the training set would largely affect the performance of the model, which words need to be extracted becomes an important problem to be solved. In this paper, we mainly consider extracting words from the description and weighting them with their appearance in the corresponding titles. In detail, we first use the TextRank algorithm proposed in [13] to extract a set of words that are considered important in its constructed text graph based on the whole content of the description. In the text graph, each node represents a word and there would be an edge if two words both appear in the same N-gram window shifting in the description. The importance of nodes in the graph is calculated as the PageRank score in the TextRank tool, and the words are ranked according to their PageRank scores.

Table 2: An Eclipse Bug Report with stack trace

Bug ID: 54804
Title: BadLocationException when deleting code containing a breakpoint
Descrtption: In my development workbench, while having a target workbench open in debug mode ... I got this: org.eclipse.jface.text.BadLocationException at java.lang.Throwable.(Throwable.java) at org.eclipse...BadLocationException.(BadLocationException.java:25) at org.eclipse...getLineNumberOfOffset(AbstractLineTracker.java) at org.eclipse...getLineOfOffset(AbstractDocument.java) at org.eclipse...breakpointAtRulerLine(AbstractBreakpointRulerAction.java) at org.eclipse...AbstractBreakpointRulerAction.determineBreakpoint (AbstractBreakpointRulerAction.java) at org.eclipse...EnableDisableBreakpointRulerAction.update (EnableDisableBreakpointRulerAction.java:48) ...

Besides ranked keywords by TextRank, we also consider retrieving keywords from the stack trace within the description. The stack trace (Table 2 shows an example) is found to be useful in helping developers locate software defects by providing potential buggy files within its logs [14]. Following [15–17], we use regular expressions (in Table 3) to extract the stack trace and retrieve the embedded code file names which contain relatively important information and are less noisy (e.g., "BadLocationException" for "org.eclipse...BadLocationException. (BadLocationException.java:25)") [14]. The words ranked by TextRank and code file names are then taken as candidate keywords that capture bug-related information. After the above two steps, we further make use of titles of historical bug reports to determine the final set of keywords for model training.

To guarantee the quality of keywords, we only consider the words that appear in the corresponding title for a description as the final keywords (note that this strategy is only applied to building training datasets, we do not require this for testing datasets). These words link a title and a description, are more domain-specific, and usually contain the main topic of a bug report. We place these keywords just in the order they appear in a title, so as to reflect their semantic relations.

Keywords Injection. The goal of Keywords Injection is to construct keywords-title pairs that are particularly designed for our controllable title generation. Unlike titles that are generally sentences with complete syntactic and semantic rules, extracted keywords are much more discrete and have a relatively less close relation compared to that of words within a title. Mixing them together without distinguishing them would mislead our transformer-based generation model during the training phase and generate unexpected titles in the inference part. Hence, we use special marks to prefix both keywords and titles to make them recognizable by KeyTitle, so as to let it make use of keywords to generate better titles. As shown in the figure, we aim to obtain [M1]KW_i[M2]T_i pairs, where M1 and M2 are set as "keywordsMark" and "titleMark", T_i represents the ith title and KW_i represents a chain of extracted keywords associated with the title. The M1 and M2 marks are very necessary as it makes the generation model distinguish keywords from titles to avoid taking them as the same thing during model training.

3.3 Controlled Generation

Due to the rapid development of deep learning, a number of pretrained languages models (PLMs) such as BERT [18] have been proposed and demonstrate their effectiveness in a number of natural language processing (NLP) tasks [18–20]. It comes with no surprise that PLMs largely dominate the research area of text generation [19–21]. PLMs are generally trained on large web datasets such as Wikipedia, Google News, which make PLMs learn much general knowledge of the world. In practice, to let model better learn more about domain-specific knowledge, researchers/practitioners would use a small dataset from their application domains to fine-tune the PLMs to achieve better performance for their down-stream tasks. Regardless of using general PLMs or fine-tuned PLMs, they generally adopt the direct Seq2Seq way (end-to-end) to generate titles from descriptions. This sometimes would make generated target results not controllable or even unpredictable especially for cases where the guidance source are not of high quality. Hence, in this paper, we propose controlled generation for titles. In other words, we do not adopt the end-to-end generation. Instead, we enforce our model to generate keywords first, then use the keywords as generation direction to guide the title generation process.

More specifically, as shown in figure 1, we achieve controlled generation by building an encoder-decoder transformer model upon titles injected with keywords (i.e., [keywordsMark]keywords[titleMark]title) as well as the descriptions. The transformer model would learn the embeddings of the descriptions and injected titles first (i.e., converting textual contents to numeric vectors of certain dimensions). After that, the embedded description is passed to the encoder. The encoder would then turn the embedded description into a sequence of continuous representations, which are then

fed into a decoder. The embeddings of titles injected with keywords would work as the output embeddings, used as the training target. The decoder receives both the output of encoder and the decoder output at previous time to predict the next token. In specific, when predicting the next token W3 for "[M1] KW1 KW2 [M2] W1 W2", the decoder will consider not only the output of encoder but also the output already generated (from [M1] to W2). When W3 is calculated, it will be appended after W2. The whole new sentence (from [M1] to W3) will be considered to calculate a new token W4. When the predicted token is the special end token (i.e., $< /s >$), the generation process stops and we then obtain a title corresponding to the input description.

With the keywords/title mark, the model would recognize that keywords and titles are two different targets and should be treated as different generation tasks. Meanwhile, as the keywords are placed before the titles, the model would generate the keywords first, then the titles. Since we use the autoregressive decoder that would take the previous output as additional input when predicting the next token, we can make sure that the knowledge of keywords would be used to guide the title generation. In this way, we make it possible to perform controllable generation of titles guided by keywords.

4 Experimental Setup

4.1 Dataset Construction

To fully understand the potential of KeyTitle in generating titles for bug reports, we test KeyTitle on three large bug datasets collected from different platforms that adopt different bug tracking systems to configure their bug reports. They are Eclipse bug reports managed by Bugzilla, Apache bug reports managed by JIRA, and iTAPE data (shared by iTAPE [6], the state-of-the-art bug title generation technique) managed by GitHub issue tracking system. Both Bugzilla and JIRA configured a bug report template with much more items than GitHub issue template. We crawl bug reports of Eclipse and Apache from their bug tracking systems. The time span of the Eclipse dataset is from Oct 2001 to May 2019, and Apache from Apr 2003 to May 2021. The total number of bug reports crawled from Eclipse and Apache bug report repositories are 567,898 and 1,038,043, respectively.

Among the collected 567,898 and 1,038,043 bug reports of Eclipse and Apache, we filtered out those whose title or description is missing as they are not applicable to model training. Furthermore, titles with fewer than 5 words or more than 15 words were also excluded. This was because existing studies reported that titles with such lengths (5-15 words) were more likely to accurately and concisely describe the key ideas of bugs; we also manually sampled and checked a few title examples and also found that titles with fewer than 5 words tended to be insufficient to contain key information, while titles with more than 15 words tended to be redundant. Removing these titles (taking up about 31% of all bug reports in Eclipse, and 40% in Apache) could help us obtain a quality-acceptable dataset to build the generation model. Given that this step aims to obtain a relatively high-quality dataset for model learning, even we missing some good titles with <5 or >15 words would not affect the validity of the built methods.

For each left bug report, we extract its description and title items. Following [6], we use several regular expressions shown in Table 3 to remove tags (if exist) in titles as our focus is on generating the descriptive text of a title. For the description item, we further replace the concrete url websites with the general string "URL" to make models understand the information type without being misguided by detailed individual url strings during model training. Bug reports whose titles contain url websites would be directly removed. For those bug reports with stack traces (taking up about 12%-21% of bug reports in Eclipse and Apache, as shown in Table 4), we only considered the file names embedded within stack traces (e.g. "BadLocationException" for "org.eclipse...BadLocationException.(BadLocationException.java:25)"), since file names are considered to be less noisy compared to the full file paths and it helps better reveal bug semantics [15]. Based on the textual contents of descriptions, keywords are then extracted by using the approach in Section 3.2.

After the above preprocessing, each bug report would be transformed into a three tuple `<the description, keywords, the title>`. These tuples constitute our datasets for model building and prediction (Figure 1 reveals the detailed use of the three elements of these tuples). We randomly split the datasets into training, validation and test sets with a ratio of 8:1:1. Table 4 provides the basic statistics of our experimental data.

Table 3: Regular Expressions

Type	**Regular Expression**
URL	`(https?|ftp)://[^\s/$.?#].[^\s]*`
"[Tag]" at Beginning	`^(\s*\[.* ?\])+`
"Tag:" at Beginning	`^.*?:`
Stack Trace	`\w+\(([\w]+)\.\w+:\d+\)` or `/(\w+)\.\w+:\d+`

Table 4: Basic Statistics of Experiment Datasets

Dataset	**Train**	**Validate**	**Test**	**Avg KW Count**	**Avg Title length**	**Avg Des length**	**% Stack Trace**
iTAPE data	267,094	33,031	33,438	2.76	9.09	124.29	1.03%
Eclipse	312,885	38,908	38,212	2.70	8.28	108.00	21.23%
Eclipse iTAPED	221,460	28,275	29,314	3.02	8.95	114.16	14.69%
Apache	489,866	62,390	60,937	2.60	9.01	107.47	18.03%
Apache iTAPED	336,244	41,263	38,604	3.01	9.46	123.68	12.52%

4.2 Model Architecture

Our KeyTitle is built on a transformer-based encoder-decoder model namely T5-base [19] by integrating keywords planning. Both the encoder and decoder consist of 12 blocks, which comprise self-attention, optional encoder-decoder attention and a feed-forward network. The output dimensionality of the feed-forward network is set as 3072, following a ReLU nonlinearity and another dense layer. All attention mechanisms have 12 heads and all other sub-layers and embeddings have a dimensionality of 768. We use the HuggingFace Transformers library[22] to implement the models.

4.3 Baseline Methods

We choose two baselines to compare the generation performance of KeyTitle. One is iTAPE, the state-of-the-art title generation technique particularly designed for software issues. The other one is the fine-tuned T5 that uses a certain number of bug reports to fine-tune the T5 model, a well-performed pre-trained model for general text-to-text NLP tasks.

iTAPE: iTAPE[6] uses a seq2seq model to generate a title based on the issue description. The model mainly includes an encoder network, a decoder network, an attention layer, and a final generator. The encoder and decoder network is Bidirectional LSTM and unidirectional LSTM respectively. To solve the OOV problem resulted from the fixed vocabulary dictionary used by seq2seq model, iTAPE adds special markers to human-named tokens (for version number and identifiers) first and then uses a copy mechanism to copy these tokens from the input to train a generation model.

T5: T5[19] is a popular pretrained model based on the transformer architecture, working on converting NLP tasks into a unified text-to-text task. The model is pretained on the C4 dataset of 750GB documents by using a BERT-sytle denoising objective. In the pretraining process, some input tokens within a sentence are corrupted, and the model is trained to reconstruct the original text from the corrupted input. This helps the model learn to understand the relationships between different tokens and how they fit together in a sentence.

We run all the experiments on a single NVIDIA 3080 (10GB). To compare with iTAPE, we preprocess the datasets we collected and run the experiments with the scripts provided by the authors. For T5 and KeyTitle, we run them with batch size of 4 in 10 epochs with learning rate 1e-5. The warmup process of Adam Optimizer is set to the first epoch.

4.4 Evaluation Metrics

Evaluation based on Typical Metrics: To evaluate the performance of our KeyTitle, we use the ROUGE-N F1 scores that are typically used for generation tasks to measure the quality of generated titles. Three ROUGE metrics, namely ROUGE-1, ROUGE-2, and ROUGE-L are considered. ROUGE-1, ROUGE-2, and ROUGE-L measure the overlap of unigrams, bigrams, and Longest Common Subsequences between generated summary and the reference summary separately. The formulas to calculate them are as follows:

$$R_{ROUGE-N} = \frac{\sum(overlapped\_N\_grams)}{\sum(N\_grams\_in\_reference\_summary)}$$
$$P_{ROUGE-N} = \frac{\sum(overlapped\_N\_grams)}{\sum(N\_grams\_in\_generated\_summary)}$$
$$F1_{ROUGE-N} = 2 \times \frac{R_{ROUGE-N} \times P_{ROUGE-N}}{R_{ROUGE-N} + P_{ROUGE-N}}$$

R(recall) and P(precision) calculate the percentage of overlapped N-grams in the reference summary and generated summary respectively. F1-score is a way to balance the trade-off between recall and precision. In general, a high F1 score indicates that the model achieves both high recall and high precision at the same time, while a low F1 score indicates that the model may have low precision, low recall, or both.

Human Evaluation: Besides ROUGE metrics, we also conduct a human evaluation to better understand the generated bug titles. We adopt a three-step approach to check whether KeyTitle would generate better titles than two baselines, i.e., iTAPE and fine-tuned T5. First, we randomly select 384 bug reports from three datasets according to their bug proportion, by following the sampling strategy in [23–25]. Such sampling could let us achieve a confidence level of 95%, with sampling error within the range of ± 5%. Within each dataset, two kinds of bug reports that are considered of high or low quality by iTAPE are separately sampled also based on their quantity proportion. The detailed numbers of the sampled bug reports from three datasets are shown in Table 5.

Then, for each selected bug report, we run KeyTitle, iTAPE, and fine-tuned T5 separately to generate a title from its description. Last, we recruit several participants to rate the quality of titles for each bug report. Each bug report would be presented as a description associated with four titles, i.e., the original title from the bug report, and three titles generated by KeyTitle, iTAPE, and fine-tuned T5. To avoid potential bias of participants towards a certain technique, we place four titles of a bug report in a random order so that the raters are not able to know which title is generated by which tool. Three participants take part in our human evaluation experiments. They are all postgraduates in software engineering with over 5 years of experience in programming. Each participant is asked to independently score the titles for 384 bug reports in terms of the following four aspects, namely Relevance, Accuracy, Conciseness, and Fluency. For each aspect, they are asked to score on a Likert scale of 1-5 for poor, better than poor, okay, better than okay, and great, respectively.

- *Relevance*. The title should contain bug-related information revealed by the bug report.

Table 5: Sampled Bug Reports for Human Evaluation

Dataset	Filtered by iTAPE	Not filtered
iTAPE data	None	78
Eclipse	21	90
Apache	52	143

- *Accuracy*. The title should be accurate with respect to the bug report or the domain knowledge.
- *Conciseness*. The title should not have redundant or unnecessary content.
- *Fluency*. The title should be grammatically correct and describe the fact fluently.

4.5 Research questions

In this work, we will investigate the following research questions (RQs) to evaluate the performance of KeyTitle.

***RQ1**: Does KeyTitle generate better titles than baselines?*

This RQ aims to provide a quantitative comparison between the performance of KeyTitle and the state-of-the-art iTAPE and fine-tuned T5 generation models with ROUGE metrics.

***RQ2**: How good are the titles generated by KeyTitle from a practitioner's perspective?*

As a method aiming to generate titles for human practitioners, it is important to know their feedback about the quality of generated titles, which could provide directions for our further improvements.

***RQ3**: How well does KeyTitle perform in generating titles directly from keywords?*

Existing studies mainly focus on generating titles from the descriptions of bug reports. Little attention is paid to exploring the scenario that generates a title using a few keywords given by end users, a task that has broad potential application values.

5 Experimental Results and Discussions

5.1 *RQ1: Does KeyTitle generate better titles than baselines?*

Goal. In this RQ, we mainly explore whether our KeyTitle can outperform the state-of-the-art iTAPE and fine-tuned T5 in terms of typical ROUGE metrics, as well as the ability to solve the OOV problem.

Experimental Setup. We compare KeyTitle with iTAPE and fine-tuned T5 on different bug datasets from Eclipse, Apache, and GitHub (the GitHub dataset is collected and shared by iTAPE) shown in Table 4. Note that within iTAPE, three heuristic rules would be applied first to filter bug reports (hence obtained the filtered Eclipse and Apache datasets) before title generation model construction. While for our KeyTitle, we has a lower requirement for the quality of bug reports that we only used the first rule of iTAPE to filter bug reports. To keep a fair comparison, we decided to only evaluate iTAPE on the filtered Eclipse and Apache datasets (which actually provides stronger baselines than that on unfiltered datasets). Furthermore, as the authors kindly shared the iTAPE tool and iTAPE data, we only run KeyTitle and fine-tuned T5 on iTAPE data and compare the results directly with their reported results of the iTAPE. For the filtered Eclipse and Apache datasets, we train T5, KeyTitle and iTAPE (with 30,000 steps) separately to obtain their corresponding performance.

Besides running three techniques on three filtered datasets, we also run fine-tuned T5 and KeyTitle on the original Eclipse and Apache datasets, in order to better understand the effects of keywords planning (as keywords planning is the major difference

between these two techniques). We train the fine-tuned T5 and KeyTitle with the same training settings, i.e. the epoch, learning rate, and warmup strategy. We checkpoint the two models at each epoch, and select the best checkpoint based on the performance of validation datasets to carry out the performance comparison between them.

Table 6: Performance Comparison between KeyTitle and Two Baselines in Terms of ROUGE Metrics.

Model	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
iTAPE data			
iTAPE	31.36	13.12	27.79
T5	33.04	14.14	30.39
KeyTitle	**34.14**	**14.61**	**31.41**
Filtered Eclipse data			
iTAPE	30.61	12.56	28.02
T5	32.68	13.8	29.82
KeyTitle	**34.10**	**14.63**	**31.13**
Filtered Apache data			
iTAPE	27.97	10.59	25.59
T5	32.33	13.46	29.48
KeyTitle	**33.07**	**13.81**	**30.19**
Eclipse data			
T5	26.52	11.51	24.82
KeyTitle	**27.76**	**12.19**	**26.06**
Apache data			
T5	26.02	11.76	24.24
KeyTitle	**27.08**	**12.29**	**25.35**

Results. Table 6 present the performance comparison results of three models on different datasets with respect to three ROUGE metrics. According to the table, we can find that both KeyTitle and fine-tuned T5 perform better than iTAPE on three filtered datasets in all three ROUGE metrics. In detail, for the shared iTAPE data and the two filtered Eclipse and Apache datasets, KeyTitle could outperform iTAPE by relatively 8.9-18.2%, 11.4-30.4%, and 13.0-18.0% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores respectively. The fine-tuned T5 outperforms iTAPE by relatively 5.4-15.6%, 7.8-27.1%, and 6.4-15.2% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores respectively. The comparison results of KeyTitle and fine-tuned T5 over iTAPE to a certain degree demonstrate the power of pre-training in the bug report title generation task (both fine-tuned T5 and KeyTitle are based on pre-trained models). This is explainable, as pre-training brings more general knowledge into a model, making the model easier to generate more fluent and reasonable titles. As a result, a simple fine-tuning operation could make the model generate titles of higher quality than a classic seq2seq model (adopted by iTAPE).

As for the comparison between KeyTitle and fine-tuned T5, we can find that KeyTitle could outperform fine-tuned T5 by relatively 2.3-4.7%, 2.6-6.0%, and 2.4-5.0% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Such an ablation study indicates that, incorporating keywords planning into the training process could indeed make KeyTitle understand better the key points and main topics of a bug report, thus generating higher-quality titles than simply fine-tuned pre-trained models with bug reports (fine-tuned T5).

Table 7: Title Examples Generated by iTAPE, fine-tuned T5 and KeyTitle

Example 1 (From iTAPE data)	**Example2 (From Eclipse data)**	**Example3 (From Apache data)**
Description: i 'm using the phofurl repository to keep my debianubuntu machines up to date. i 've noted that v verid40 1.3.1 verid0 was released today - for syncthing, discovery and relay, but the apt.syncthing.net repository seems to be missing the last two. phofnewline phofnewline... **Original Title:** apt.syncthing.net missing 1.3.1 release (discovery & relay)	**Description:** i think it would be useful if _.flatten flats anything that is iterable, which usually means they implement symbol.iterator. the thing is i am currently migrating some of the jquery parts of a web to a native js alternative. the problem comes when i want to migrate something like $elements.find(input)... **Original Title:** should _.flatten flat anything iterable ?	**Description:** build 20020307 1. create the following compilation unit: public class a public static interface c void foo(); 2. open it in a java editor 3. select all 4. paste in the following content: public class... 5. revert observe: syntax coloring has been lost: every character is black... **Original Title:** Syntax coloring lost on revert
iTAPE: v 1.3.1 does n't list the ? repository **T5:** syncthing 1.3.1 missing from apt.syncthing-.net **KeyTitle:** missing 1.3.1 release from apt.syncthing.net repository	**T5:** _.flatten should work with anything iterable **KeyTitle:** _.flatten should flats anything that implements symbol.iterator	**T5:** Syntax coloring lost after revert **KeyTitle:** Syntax coloring lost when reverting in java editor

Case Study & Analysis. The value of ROUGE metrics demonstrated the effectiveness of our KeyTitle over baselines from a quantitative perspective. To better understand the advantages or limitations of KeyTitle. We conducted a qualitative comparision among the original titles, the titles generated by KeyTitle and the other two baselines (i.e., iTAPE and fine-tuned T5) on the sampled datasets of 384 bug reports shown in Table 5. Our analyis includes two parts. The first part is to manual

compare the quality of titles generalted by KeyTitle and the original titles (i.e., the ground truth titles provided in the 384 bug reports). The second part is to check the titles generated by KeyTitle and the two baselines. Details are as below.

We follow a two-step approach to compare the quality of titles generated by KeyTitle and that of the original titles provided in the bug reports. Two authors (acting as participants) get involved in analyzing the quality of the sampled 384 bug reports. In the first step, they would randomly select 100 bug reports, analyze their content and the associated titles, and then obtain the initial quality categories. Specifically, for each bug report, the two participants would read the description first to gain an overall understanding of the reported bug. Then, the corresponding original title and generated title by KeyTitle are investigated to determine whether one has higher (or equal) quality than the other. In terms of quality, the participants would mainly consider four aspects, i.e., Relevance, Accuracy, Conciseness and Fluency (i.e., the four metrics used in human evaluation in Section 5.2) of a title, with more focus on the Relevance and Accuracy (a title should correctly reveal the reported bug first). Based on their comprehensive judgment, the comparison results of titles generated by KeyTitle and the original titles are divided into three quality categories, namely higher-quality, lower-quality, and equal-quality, representing that the titles generated by KeyTitle have higher, lower and equal quality than the original titles, respectively. For each category, the two participants discussed together and further summarized several subcategories showing how exactly the titles generated by KeyTitle are better or worse than the original titles. This provides valuable insights for improving KeyTitle and exploring potential practical applications. Table 8 shows the initial concluded subcategories.

In the second step, after getting the initial quality categories from 100 sampled bug reports, the two participants then independently analyzed the remaining 284 bug reports by following the same process (as in the first step) and placed each bug report into each subcategory in Table 8. If the subcategories could not cover the bug reports, they could add new subcategories. Actually, no extra subcategories are added after they finish checking the 284 bug reports. Hence, the categories shown in Table 8 are the final category results of quality comparison. The Fleiss' Kappa [26] is used to measure the agreement between the two participants; and the overall Kappa value was 0.76, indicating a substantial agreement according to [27]. For any disagreement related to categorizing bug reports, the two participants would discuss together to reach a common decision.

For the comparison of titles generated by KeyTitle and the original titles, we find that about 49% of the titles generated by KeyTitle are of equal quality as the original ones; 36% show improvements, and 15% are of worse quality than the original titles. The improvements of 36% titles over original titles can be concluded into three categories. First, 12% of titles tend to contain less redundant information (as Example 1s in Table 7 and 8 show). Pre-trained on large-scale datasets, KeyTitle is still able to generate natural and smooth titles when fine-tuned on downstream tasks. Therefore, redundant information that is difficult to understand is naturally filtered out. Second, 14% of titles tend to point out which specific function in particular is under discussion (as Example 2s in Table 7 and 8 show). When users search for relevant information, they will pay more attention to the specific error function. This improvement can

Table 8: Quality Comparison Results of Titles Generated by KeyTitle and the Original Titles

Advantages Over Original Titles (36%)		
Less Redundant Information (12%)	**More Specific Function (14%)**	**Detailed Program Context (10%)**
Example 1: **Description:** ... scan my repo and look in the generated nexus index properties file there is always has the wrong timestamp:...nexus .index .time = 19700101010000 .000 + 0100... **Original Title:** nexus .index .time always set to 19700101010000.000 + 0100 **KeyTitle:** nexus .index .time has wrong timestamp	**Example 2:** **Description:** it seems that extracting large zip packages is much slower than with java.util.zip. in certain tests, the extraction time of 2 large packages was almost 1 hour more with vfs. ... **Original Title:** extraction process is unefficient with large packages **KeyTitle:** extracting large zip packages is much slower than with java.util.zip	**Example 3:** **Description:** when starting axis2server, it does not return output to the server console. steps to reproduce : 1 . start axis2server in axis2-1.3-rc2 2. check the console. nothing will be shown there... **Original Title:** axis2server console output is not shown **KeyTitle:** axis2server does not return output to server console
Disadvantages Against Original Titles (15%)		**Basically Equal Quality (49%)**
Repetitive Words (3%)	**Unnecessary Information (12%)**	**Basically Equal Quality (49%)**
Example 4: **Description:** this bug was imported from another system and requires review from a project committer before some of the details can be marked public. for more information about historical bugs, please read... **Original Title:** rte when tabbing to open a menu and then mousing down elsewhere in the application **KeyTitle:** if you click on a button, you get a tuple, a tuple, a tuple...	**Example 5:** **Description:** ## summary: when using git module, if ssh ://git@192 .168 .1 .89: 1357/, the host check process breaks. steps to reproduce: 1. have a git repo available over ssh on port 22 2. have a working git checkout with ssh: //git @ fqdn/... **Original Title:** git module indicates unknown hostkey when repo has a port **KeyTitle:** git module fails with ssh ://git @ 192.168.1.89: 1357/	**Example 6:** **Description:** ...requires a try/catch block around the executequery. if you left click on the error in the left margin and select either add throws or add try/catch block, eclipse deletes the import java.lang.system!... **Original Title:** surround with try/catch or add throws declaration deletes import java.lang.system; **KeyTitle:** add throws or add try/catch block deletes import java.lang.system

reduce the time for searching and improve productivity. At last, 10% of titles tend to point out the program context where the issue occurred (as Example 3s in Table 7 and 8 show). In such cases, the basic background of issues is directly given. Such an improvement allows developers to quickly find relevant bug reports and locate bugs.

Besides the above-mentioned improvements, we further checked the titles generated by KeyTitle that show worse quality than the original ones, with the hope of finding possible ways to improve KeyTitle in the future. We summarized mainly two scenarios that KeyTitle failed to perform well. Specifically, we found that 3% of titles generated by KeyTitle may contain repetitive words. As Example 4 in Table 8 shows, KeyTitle generated "a tuple" in a row. This inspires us to reduce the duplication in the generated title during the training process and improve the diversity and quality of the generated title. Meanwhile, in 12% of cases, titles generated by KeyTitle may contain unnecessary information. As Example 5 in Table 8 shows, although the ssh command is important as it shows the port information for developers to better locate the bug, it is not necessary to appear in the title. The strange symbols (like "@" or "#") may drive the attention of model away, thus making the model generate unusual information.

As for the comparison of KeyTitle over iTAPE and the fine-tined T5, several typical cases in Table 7 are presented to understand the advantages of our KeyTitle. For Example 1, we can find that, the version number 1.3.1 is generated successfully by all three models, proving their ability in generating version numbers. However, iTAPE still miss the token "apt.synching.net" despite it also paid much attention to the identifiers. This is predictable as the seq2seq model adopted by iTAPE uses a fixed vocabulary dictionary, which may not adapt well to various software development activities with new tokens continuous appearing. Hence, the OOV problem may always exist for iTAPE.

In contrast, the fine-tuned T5 and KeyTitle successfully generate the domain-specific token "apt.synching.net". The possible reasons are as follows. T5 uses SentencePiece[28] to tokenize a sentence into subwords and neural machine translation to deeply consider the relationship between subwords. Hence even for rarely-observed or newly appeared human-named-tokens that are composed of numbers, characters, or special symbols, T5 will cut them into subparts and consider their relationship during model training and generate text in a deeper perspective. In this sense, it is much less likely for T5 to suffer the OOV problem, so does KeyTitle built upon T5. Also, we notice that after training on the large dataset, KeyTitle learns to choose more suitable keywords. In the original title, "discovery & relay" are placed in a pair of parentheses, which is not so normative in a bug report. After training, KeyTitle learns to discard this phrase and generates a more succinct title. This demonstrates that after continuously learning from data, the model can better determine keywords based on semantics.

In Example 2, the original title is asking a question. However, from the description, we can know that the developer is actually trying to make a suggestion. Both KeyTitle and fine-tuned T5 use declarative sentences as titles to make the statement of the problem more objective. Compared to fine-tuned T5, KeyTitle further generates "that implements symbol.iterator", which clearly points out the function. In Example 3, the title generated by KeyTitle also conveys important information "java editor". "Syntax

coloring" is usually used in a text editor or integrated development environment. The usage scenarios are quite different and the original title does not indicate the difference. The title generated by KeyTitle allows developers to quickly determine whether the bug report is relevant based on their actual scenarios. As for fine-tuned T5, it generates a title that is almost the same as the original title without pointing out the scenario like Keytitle. This demonstrates that, with keywords planning, KeyTitle tends to better understand the deep semantics within the text and hence can generate better titles.

> KeyTitle outperforms both iTAPE and fine-tuned T5 in all three ROUGE metrics; With keywords planning, our pretrained model based KeyTitle could better grasp the main points of a bug report, learn more deep semantics within the text, and hence generate higher-quality bug titles.

5.2 *RQ2: How good are the titles generated by KeyTitle from a practitioner's perspective?*

Goal. We have evaluated the performance of our KeyTitle based on typical ROUGE metrics in RQ1. In this RQ, we aim to evaluate the quality of generated titles based on human evaluation. This could give us both a quantitative and qualitative view about the quality of titles generated by KeyTitle over other baselines.

Experimental Setup. As introduced in Section 4.4, we randomly select 384 bug reports from three datasets according to their bug proportion. Then, for each selected bug report, we run KeyTitle, iTAPE and fine-tuned T5 separately to generate a title from the description. After that, three participants are asked to independently rate these titles together with original titles on a Likert scale of 1-5 on four aspects, namely Relevance, Accuracy, Conciseness and Fluency. For each aspect, an average score of three scores from three raters is calculated as the final score for each title. We collect these final scores and conduct evaluations based on them.

Results. Fig 2 shows the average scores of 384 titles on four evaluation aspects based on human evaluation. We use Fleiss' Kappa [26] to calculate the inner agreement between annotators. The Fleiss' Kappa value is 0.71, indicating a substantial agreement according to [27]. From the figure, we can find that KeyTitle achieves better results in all four aspects than fine-tuned T5 and iTAPE, which is consistent with the findings of evaluation based on ROUGE metrics in RQ1. Specifically, when considering Relevance and Accuracy, KeyTitle has obvious advantages compared with other methods. This phenomenon demonstrates that with keywords planning, KeyTitle could better capture the key points in the description, and its wording is more in line with software engineering conventions.

As for the Conciseness and Fluency aspects, KeyTitle also performs better than other methods. This again indicates that with pre-trained knowledge, KeyTitle can better generate smooth, easy-to-understand titles.

Noticeably, we find that the average rating scores towards original titles (i.e., the ones filled by bug reporters in the bug reports) are all smaller than that of titles generated by title generation tools (note that raters do not know the sources (from

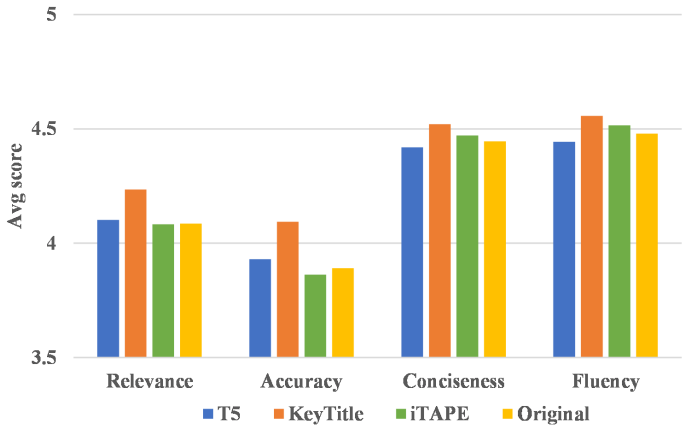

Fig. 2: Average Likert Scores based on Human Evaluation

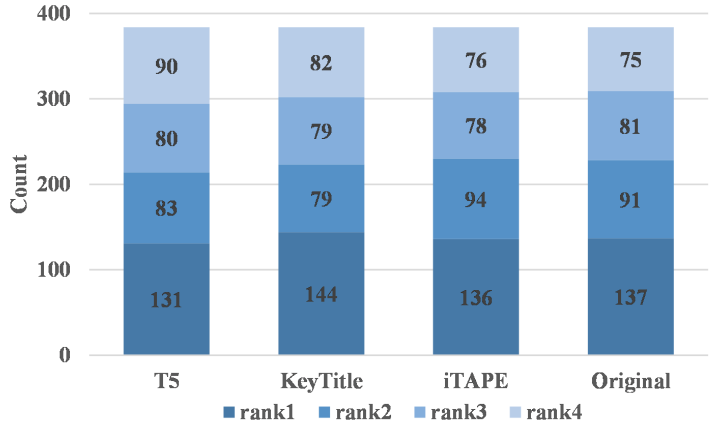

Fig. 3: Title Rank Distributions of Different Techniques

a tool or bug reporter) of titles). This to some extent further justifies the need to develop automated methods for title generation.

Besides considering individual aspects (in figure 2), for each method, we further consider the overall quality of its generated title by averaging four aspect scores. For example, suppose the scores of Relevancy, Accuracy, Conciseness, and Fluency are 3,4,5,3 respectively for a title generated by iTAPE, then their average value 3.75 (3+4+5+3/4=3.75) is used to measure the overall quality of the title for iTAPE. For each sampled bug report (384 in total), we could obtain four average scores corresponding to four methods , i.e., KeyTitle, fine-tuned T5, iTAPE, and the original title from bug reporters (noted as Original). Then we rank these four values and place the corresponding rank to the rank bag of each method. For instance, suppose the average

scores for a bug report are 4.2, 3.8, 4.5, 3.7 for KeyTitle, fine-tuned T5, iTAPE, and Original, then rank2, rank3, rank1, rank4 are put into their rank bag respectively. The rank bag of each method would have 384 ranks for 384 sampled bug reports. Then we draw their rank distributions to better observe how raters perceive the quality of each title on the whole.

Fig 3 shows the distribution of ranks for each method. From the figure, we can learn that KeyTitle gets largest rank1 among compared methods. iTAPE achieves a similar result with that of original titles. Fine-tuned T5 are least preferred to human raters with smallest rank1 count. This may be because fine-tuned T5 learns relatively shallow knowledge without keywords planning. Hence for descriptions composed of different components, it is sometimes hard for the model to take consideration into every aspect clearly during training. For rank2 count, both iTAPE and Original outperform fine-tuned T5 and KeyTitle.

> Titles generated by KeyTitle achieve the highest scores in terms of Relevance, Accuracy, Conciseness, Fluency in manual evaluation. Besides, KeyTitle obtains the largest number of bug reports for which the titles generated by KeyTitle are considered to have the best quality on the whole.

5.3 *RQ3: How well does KeyTitle perform in generating titles directly from keywords?*

Goal. KeyTitle is designed to generate a title from a description by generating keywords first and then using these words to guide title generation. This reveals that KeyTitle actually could be used to conduct two tasks, one is to generate titles from descriptions (i.e., description2title task), and the other one is to generate titles directly from some keywords (i.e., keywords2title task). Both two tasks have great potential application values. In RQ1, we have demonstrated the effectiveness of KeyTitle for the description2title task. In this RQ, we make the first attempt to explore how good the generated titles would be if we only feed KeyTitle a set of keywords.

Experimental Setup. To make KeyTitle generate a title based simply on keywords, we need to first build keywords2title datasets for model training. We obtain such datasets from corresponding description2title datasets, where for each bug report, the keywords extracted from its description is taken as model input and the title of the bug report is taken as the output. Like description2title task, we also use the typical ROUGE metrics to evaluate the model performance. Since there is no other studies that attempt to generate bug titles from keywords, we use the description2title results of KeyTitle as baseline to evaluate the quality of generated titles. We think this is reasonable as both two tasks target at generating titles; we decide not to take iTAPE or fine-tuned T5 as baselines since KeyTitle outperformed them in the description2title task in all ROUGE metrics already (RQ1).

Results. Table 9 presents the performance comparison results. From the table, we can find that KeyTitle performs better in keyword2title task than in description2title task in all three ROUGE metrics. In detail, for iTAPE, Eclipse, and

Table 9: Keyword2title vs Description2title

Dataset	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
iTAPE data (description2title)	34.14	14.61	31.41
iTAPE data (keywords2title)	**43.64**	**17.57**	**42.76**
Eclipse (description2title)	27.76	12.19	26.06
Eclipse (keywords2title)	**40.00**	**17.42**	**39.17**
Apache (description2title)	27.08	12.29	25.35
Apache (keywords2title)	**41.04**	**16.51**	**39.95**

Apache datasets, KeyTitle could substantially outperform by absolutely 9.50-13.96%, 2.96-5.23%, and 11.35-14.6% in keywords2title task than in description2title task in ROUGE-1, ROUGE-2, and ROUGE-L. These results demonstrate the generality of KeyTitle with being able to adapt to the description2title and keywords2title tasks well. This also validates the power of keywords planning. As through keywords planning, KeyTitle is able to generate a title of relatively good quality with simply a few keywords.

The performance difference between the keywords2title and description2title tasks may lie in the amount of information related to the bug provided to the model. For the keywords2title task, we directly provide the model with the keywords most relevant to the defect report, so the model can quickly grasp the key points of the generated content. What the model needs to consider more may be various prepositions or the tone of the title, etc. However, as for the description2title task, although we have provided keywords as a guide, the detailed description of the bug report is generally long and may contain noisy information [15] that biases the generation direction of bug titles.

> KeyTitle shows great potential in generating qualified titles from keywords. It performs much better in the keyword2title task than in description2title task.

6 Discussion

6.1 GPTs in title generation

In Section 5, we compared KeyTitle with iTAPE and fine-tuned T5, which were all particularly designed for the bug title generation task. To better understand the potential of our KeyTitle, we further explored whether it would outperform some powerful large language models like GPT-4, which are designed not for particular but general generation tasks. Specifically, we chose to compare KeyTitle with GPTs (proved to perform best in various generation tasks), namely GPT-4 (which requires payment for API use based on the number of input and output tokens) and GPT-3.5 (which is free for anyone to use). Since the pay mechanism of GPT-4 makes it unrealistic for us to use the whole large test datasets for experiments, we decided to use the same dataset in the human evaluation in RQ2. This dataset consists of 384 bug reports that are randomly sampled from the whole dataset with a confidence level of 95% and sampling

error within the range of ± 5% (dataset details in 4.4). We add a prompt "Generate a title for the bug report in 5-15 words:" before every bug report description to make sure GPTs understand the title generation task and generate proper titles. Then, we ran GPT-3.5 and GPT-4 on the dataset to obtain the titles for the 384 bug reports. We found that some generated titles started with a prefix (i.e., "Bug report:"). We removed these prefixes for a fair comparison during evaluation. After that, we used the same ROUGE metrics and human evaluations to measure their quality and compared with the results of KeyTitle. The comparison results are as follows.

Table 10: Performance Comparison between Models in Terms of ROUGE Metrics on Human Evaluation Dataset.

Model	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
KeyTitle	**33.70**	**15.31**	**31.12**
GPT-3.5	28.34	9.81	25.00
GPT-4	24.48	7.27	21.78

Table 11: Average Likert Scores of KeyTitle and GPTs based on Human Evaluation

Models	Relevance	Accuracy	Conciseness	Fluency
KeyTitle	**4.23**	**4.09**	**4.52**	4.56
GPT-3.5	4.17	3.89	4.37	4.65
GPT-4	4.15	3.88	4.42	**4.71**

GPTs vs. KeyTitle in ROUGE metrics: Table 10 presents the ROUGE results of GPT4, GPT-3.5 and KeyTitle. From the table, we can see that KeyTitle performs best among the three techniques. It achieves 33.70, 15.31, and 31.12 in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1 Scores, respectively. For GPT-3.5, the three ROUGE values are 28.34, 9.81, and 25.00 respectively. GPT-4 performs worst, with ROUGE-1, ROUGE-2, and ROUGE-L F1 Scores being 24.48, 7.27, and 21.78, separately. The relative improvement of KeyTitle against GPT-3.5 could reach 18.9%, 56.1%, and 24.5% in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1 Scores. The leading advantage of our model is even more obvious when compared with GPT-4, with relative ROUGE improvements of 37.66%, 110.5%, and 42.88%, respectively, in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1 Scores. These results illustrate that our KeyTitle trained specifically for title generation performs significantly better than general-trained GPTs.

GPTs vs. KeyTitle in human evaluation: Table 11 shows the average results of KeyTitle and GPTs on four evaluation aspects based on human evaluation. From this table, we can find that, on the whole, KeyTitle still generates better titles than GPT3.5 and GPT-4 based on human evaluation (except on the Fluency metric that KeyTitle obtains slightly smaller values than GPTs). Specifically, KeyTitle obtains the highest scores of 4.23, 4.09, and 4.52 in terms of relevance, accuracy, and conciseness,

compared to that of GPT-3.5 (being 4.23, 4.09, and 4.52) and GPT-4 (being 4.15, 3.88, and 4.42), respectively. These results indicate that KeyTitle is more capable of generating more concrete and accurate bug titles than GPTs. KeyTitle only performs slightly worse than GPTs in the Fluency aspect, where GPTs obtain 4.71 (for GPT-4) and 4.65 (for GPT-3.5) scores while KeyTitle gets 4.56. These results demonstrate that with much more pre-training data and larger model architecture, GPT-3.5 and GPT-4 may generate more natural and fluent titles than KeyTitle.

Table 12: Title Examples Generated by KeyTitle, GPT-3.5 and GPT-4

Example 1	**Example2**
Description: ...This class provide an instance implementation of StageContext and maybe extended to provided additional functionality of included in other classes wishing to in the StageContext Interface @author sheldon / public class BaseContext implements StageContext // Map of DownstreamFeeder having the Stage as key private final Map Stage, Feeder feeders;... **Original Title:** Add a base Implementation for StageContext	**Description:** when pooled bytebuffer s are used in conjunction with a protocolcodecfilter, buffers passed to protocolencoderoutput .write ( ) do not get released back to the pool in all circumstances, causing unnecessary memory allocation . the following patch appears to resolve this issue in a simple filter chain, but it should be verified that there is no case where this change... **Original Title:** bytebuffer passed to protocolencoderoutput.write (bytebuffer) does not get released back to the pool
KeyTitle: [keywords] instance implementation StageContext [title] Add an instance implementation of StageContext **GPT-3.5:** Basecontext Class Implementation for Pipeline Operations **GPT-4:** Code Integration Bug Report: File Compliance and Functionality Issues in BaseContext StageContext Implementation	**KeyTitle:** [keywords] protocolencoderoutput.write() pooled bytebuggers [title] buffers passed to protocolencoderoutput.write () do not get released back to pooled bytebuffer s **GPT-3.5:** Pooled ByteBuffers Not Released in ProtocolCodecFilter **GPT-4:** Memory Allocation Bug Report: Unreleased Pooled ByteBuffers in Conjunction with ProtocolCodecFilter

Case Study & Analysis. We further investigate the low-quality titles generated by GPT-3.5 and GPT-4. Table 12 shows typical title examples from human evaluation dataset generated by KeyTitle, GPT-3.5 and GPT-4. We find that there are three main reasons leading to the generation of low-quality titles of GPTs. First, GPTs may produce hallucinations(inconsistent with facts)[29] when generating titles. Second, the titles generated by GPTs sometimes summarize the issue too broadly and thus barely contain key information. At last, although GPTs acquire much general knowledge, they

may still miss the keywords when applied to domain-specific tasks and thus generate off-topic titles. We discuss these issues in detail as follows.

In Example 1, we find that GPT-3.5 and GPT-4 summarize the bug report too broadly and sometimes may produce hallucinations[29]. The original title asks to "Add a base Implementation for StageContext". It is a request for additional functionality. As for KeyTitle, we can see that it chooses "instance implementation StageContext" as keywords and thus generates content around these keywords (with the help of the autoregressive mechanism it adopted) and obtains a relatively good title. For GPT-3.5, it produces hallucinations when generating this title. The "Basecontext" and "Pipeline Operations" are not the key information in the bug report but GPT-3.5 forcibly combines them. We cannot find a clear statement in the description corresponding to the title generated by GPT-3.5. The hallucination may place a heavy burden on users to check the correctness of the generated title. The key point about "Add a base Implementation" is also missed by GPT-3.5. For GPT-4, it just points out the bug report has "File Compliance and Functionality Issues", which is sort of too broad for users to find out what is really causing the issue. Users still have to go through the bug report to get the specific information they want. An ideal title is expected to precisely and concisely summarize the key points of a bug. Too vague titles like the one generated by GPT-4 would be of little help in understanding and locating the bugs.

In Example 2, GPT-3.5 also seems to produce hallucinations. In this bug report, we can find that the pooled bytebuffer is not released properly while using a protocolcodecfilter at the same time. The bytebuffer should be released in the "pool", not the "ProtocolCodecFilter" as GPT-3.5 states. This is inconsistent with facts and thus causes factual inconsistency[29]. Although it is fluent to read, it contains completely wrong information. The title generated by GPT-4 does not make the mistake. However, it is still not concrete enough as users have to go through the bug report to find out what function is really causing the issue. On the contrary, KeyTitle captures the key information as it chooses "protocolencoder-output.write() pooled bytebuggers" as keywords. The title generated by KeyTitle demonstrates the meaning of the original title quite well.

As a base large language model trained for general purposes, the GPTs may lack important domain knowledge when applied to a specific downstream task (like the bug title generation task). Such a lack may make GPTs produce bug titles with hallucinations[29]. Meanwhile, as designed, what GPTs should output is greatly affected by the evaluators. The reinforcement learning from human feedback would make GPTs tend to generate relatively general and hence less error-prone results that are welcomed by the public[30, 31], which is not in line with our expectations for ideal bug titles.

6.2 Threats to validity

One threat to validity comes from the datasets. All experimental datasets in this study are all collected from open platforms. We cannot guarantee that our findings are applicable to other open source projects or closed industrial projects. However, considering that all bug reports are collected from famous open source foundations

and are managed by different bug tracking systems, we believe that the popularity and diversity of bug data could still shed lights on the effectiveness of our KeyTitle in practice. To enhance understanding of our KeyTitle's applicability, we appreciate any replication studies on proprietary projects or other open source products.

Another threat comes from the metrics used to evaluate KeyTitle. Despite we adopted ROUGE metrics that are commonly used in title generation task, we have to admit that they may not fully capture the nuances of title quality in practical software development scenarios. Hence, we conducted extra human evaluations on generated titles by KeyTitle and other baselines, to provide a more comprehensive assessment of the relevance and utility of the generated titles by KeyTitle and other baselines.

Furthermore, a possible threat also arises related to human evaluation. That is, on manual evaluation on the quality of titles generated by different techniques, participants may have a certain preference for the style of the generated titles, which may cause subject bias in rating titles. To prevent the bias, we write a detailed tutorial for them to read before rating; meanwhile, we randomly shuffle the titles so that they can not figure out which title is generated by which technique. The Fleiss' Kappa value of 0.71 also indicates a substantial agreement among the rating participants.

As a title generation task, we fully understand that it is fundamental to ensure the quality of datasets used for both constructing and fairly evaluating our KeyTitle and other baselines. Towards this, we determined data selection criteria based on the commonly accepted strategy proposed by Chen et al.[6], and performed human evaluation on the quality of the original titles in bug reports that work as the ground truth. Our human evaluation indicated a quality-acceptable dataset. All preprocessing steps related to dataset preparation are also detailed in the Section 4.1 Dataset Construction. In the future, we plan to analyze the impact of different preprocessing steps on the study findings to further help us understand the robustness of our methodology and the reliability of the conclusions drawn in this study.

7 Related Work

Study on bug report titles. As the summary of a bug report, some researchers try to make use of titles to better understand and manage bug reports. For example, Chaparro et al.[32] and Mills et al.[33] take the title as an important text feature of a bug report for better text retrieval-based bug localization. Budhiraja et al.[34] and Isotani et al.[35] use different word embedding models to vectorize the title and description of each bug report to conduct duplicate bug report detection. Lee et al.[36] concatenate the titles and descriptions of bug reports as the input of PLM model and then train classifiers to do bug triage. Mani et al.[37] use a deep bidirectional recurrent neural network with attention to learn syntactic and semantic features from title and description for better triaging bug reports. Zhou et al.[38] develop an automatic vulnerability identification technique for which features related to the titles are also considered during model building. These studies demonstrate the importance of titles in various downstream tasks of bug reports. The importance of titles and their varied quality motivates our work to automatically generate high-quality titles for bug reports submitted by end users.

Automated software artifacts generation. There have been a series of studies on automated generation of different software artifacts, such as bug report summary, pull request titles, commit messages, etc [6, 10, 39–41]. For bug report summarization, Rastkar et al. [42] train a summarizer to extract important sentences from the description of a bug report as the summary for the bug. Li et al.[43] utilize a deep-learning based and unsupervised method, while Liu et al.[39] use a stepped auto-encoder network with evaluation enhancement, to do better summarization. These works all aim to generate a summary of a bug report, but adopt the extractive approach, i.e., extracting original important sentences from the description as a summary. Unlike them, we adopt the abstractive summarization way to automatically generate a new one-sentence summary for the bug report as its title. Chen et al.[6] propose iTAPE to generate issue titles, by using three heuristic rules to filter issue data first and then adopt a seq2seq model to generate issue titles. Ma et al.[44] propose a deep attention-base summarization model. The model uses RoBERTa encoder to better capture contextual semantic information, stacked transformer decoder to generate titles and the copy mechanism to handle rare token problem. Compared to their study, we emphasize the necessity of considering the information structure and the importance of keywords.

Besides bug report summary/title, Zhang et al.[45] fine-tune BART to automatically generate pull request title. Fang et al.[10] use a hybrid attention network for pull request description generation. Liu et al.[46] utilize commit messages and added source code comments to also generate pull request descriptions. Jiang et al.[47] and Dong et al.[40] aim to automatically generate commit messages to help developers better understand code changes without digging into detailed implementations. Wang et al.[48] propose an approach, considering both the structual information of crash traces and the knowledge of crash-causing bugs to summarize solutions automatically. Xie et al.[49] and Ahmed et al.[41] focus on generating summarization from source code pieces to assist developers in understanding code and reduce documentation workload. Jiang et al.[50] propose a deep learning based approach to generate release notes according to pull requests, including both the change entries and change category generation. Unlike their studies, we focus on a rather different generation task, i.e., aiming to generate a concise and precise title for a bug report.

8 Conclusion and Future Work

In this paper, we propose a technique KeyTitle to automatically generate bug report titles. KeyTitle formulates title generation as a one-sentence summarization task. It incorporates keywords planning into the training process, by enforcing the model to generate keywords first, then output the titles. Experimental results show that, KeyTitle could generate titles of higher quality from descriptions than both iTAPE and fine-tuning models in terms of ROUGE metrics. During our manual evaluation, titles generated by KeyTitle also achieve higher scores in terms of relevance, accuracy, conciseness, and fluency than baselines. As a complement, a comparison between KeyTitle and general large language models like GPTs is also conducted. The results further validate the effectiveness of our KeyTitle in the specific bug title generation task. Besides the description2title task, KeyTitle also shows great potential in the

keyword2title task, where users can simply input a few keywords, and then the model will generate a qualified title for him/her.

To further improve the performance of title generation, we plan to consider using more prompts to do keywords planning. We may consider better use of code snippets; we will also try to investigate better keyword-extracting mechanisms in our future work.

Declarations

Funding This work was supported by the National Natural Science Foundation of China(No.62002161), the Fund of Prospective Layout of Scientific Research for NUAA(Nanjing University of Aeronautics and Astronautics), Scientific Research Foundation for the Introduction of Talent for NUAA.

Conflict of interest The authors declare no competing interests.

Authors' contributions Qianshuang Meng and Weiqin Zou wrote the main manuscript. Biyu Cai collected datasets. Jingxuan Zhang helped polish the paper. All authors reviewed the manuscript.

Code and data availability We share our replication package at https://github.com/mengqianshuang/KeyTitle.

References

[1] Davies, S., Roper, M.: What's in a bug report? In: Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10 (2014)

[2] Chaparro, O.: Improving bug reporting, duplicate detection, and localization. In: 2017 IEEE/ACM 39th International Conference on Software Engineering Companion (ICSE-C), pp. 421–424 (2017). IEEE

[3] Wahono, R.S.: A systematic literature review of software defect prediction. Journal of software engineering **1**(1), 1–16 (2015)

[4] Xuan, J., Jiang, H., Ren, Z., Zou, W.: Developer prioritization in bug repositories. In: 2012 34th International Conference on Software Engineering (ICSE), pp. 25–35 (2012). IEEE

[5] Fiaz, A., Devi, N., Aarthi, S.: Bug tracking and reporting system. arXiv preprint arXiv:1309.1232 (2013)

[6] Chen, S., Xie, X., Yin, B., Ji, Y., Chen, L., Xu, B.: Stay professional and efficient: Automatically generate titles for your bug reports. In: Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering. ASE '20, pp. 385–397. Association for Computing Machinery, New York, NY, USA (2021). https://doi.org/10.1145/3324884.3416538 . https://doi.org/10.1145/3324884.3416538

[7] Zhang, T., Irsan, I.C., Thung, F., Han, D., Lo, D., Jiang, L.: itiger: an automatic issue title generation tool. In: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 1637–1641 (2022)

[8] Lin, H., Chen, X., Chen, X., Cui, Z., Miao, Y., Zhou, S., Wang, J., Su, Z.: Genfl: Quality prediction-based filter for automated issue title generation. Journal of Systems and Software **195**, 111513 (2023)

[9] Sutskever, I., Vinyals, O., Le, Q.V.: Sequence to sequence learning with neural networks. Advances in neural information processing systems **27** (2014)

[10] Fang, S., Zhang, T., Tan, Y.-S., Xu, Z., Yuan, Z.-X., Meng, L.-Z.: Prhan: automated pull request description generation based on hybrid attention network. Journal of Systems and Software **185**, 111160 (2022)

[11] Qiu, X., Pei, H., Yan, H., Huang, X.: A concise model for multi-criteria chinese word segmentation with transformer encoder. arXiv preprint arXiv:1906.12035 (2019)

[12] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)

[13] Gage, P.: A new algorithm for data compression. C Users Journal **12**(2), 23–38 (1994)

[14] Wong, C.-P., Xiong, Y., Zhang, H., Hao, D., Zhang, L., Mei, H.: Boosting bug-report-oriented fault localization with segmentation and stack-trace analysis. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 181–190 (2014). IEEE

[15] Rahman, M.M., Roy, C.K.: Improving ir-based bug localization with context-aware query reformulation. In: Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering. ESEC/FSE 2018, pp. 621–632. Association for Computing Machinery, New York, NY, USA (2018)

[16] Bettenburg, N., Premraj, R., Zimmermann, T., Kim, S.: Extracting structural information from bug reports. In: Proceedings of the 2008 International Working Conference on Mining Software Repositories. MSR '08, pp. 27–30. Association for Computing Machinery, New York, NY, USA (2008). https://doi.org/10.1145/1370750.1370757

[17] Moreno, L., Treadway, J.J., Marcus, A., Shen, W.: On the use of stack traces to improve text retrieval-based bug localization. In: 2014 IEEE International Conference on Software Maintenance and Evolution, pp. 151–160 (2014). https:

//doi.org/10.1109/ICSME.2014.37

[18] Devlin, J., Chang, M.-W., Lee, K., Toutanova, K.: Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint arXiv:1810.04805 (2018)

[19] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., Liu, P.J.: Exploring the limits of transfer learning with a unified text-to-text transformer. The Journal of Machine Learning Research **21**(1), 5485–5551 (2020)

[20] Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., Stoyanov, V.: RoBERTa: A Robustly Optimized BERT Pretraining Approach (2019)

[21] Floridi, L., Chiriatti, M.: Gpt-3: Its nature, scope, limits, and consequences. Minds and Machines **30**, 681–694 (2020)

[22] Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., *et al.*: Transformers: State-of-the-art natural language processing. In: Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, pp. 38–45 (2020)

[23] Jiang, Y., Liu, H., Zhang, Y., Niu, N., Zhao, Y., Zhang, L.: Which abbreviations should be expanded? In: Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, pp. 578–589 (2021)

[24] Pascarella, L., Bacchelli, A.: Classifying code comments in java open-source software systems. In: 2017 IEEE/ACM 14th International Conference on Mining Software Repositories (MSR), pp. 227–237 (2017). IEEE

[25] Zhang, J., Wang, X., Zhang, H., Sun, H., Liu, X.: Retrieval-based neural source code summarization. In: Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering, pp. 1385–1397 (2020)

[26] Fleiss, J.L., Levin, B., Paik, M.C., *et al.*: The measurement of interrater agreement. Statistical methods for rates and proportions **2**(212-236), 22–23 (1981)

[27] Landis, J.R., Koch, G.G.: The measurement of observer agreement for categorical data. biometrics, 159–174 (1977)

[28] Kudo, T., Richardson, J.: Sentencepiece: A simple and language independent subword tokenizer and detokenizer for neural text processing. arXiv preprint arXiv:1808.06226 (2018)

[29] Huang, L., Yu, W., Ma, W., Zhong, W., Feng, Z., Wang, H., Chen, Q., Peng, W., Feng, X., Qin, B., et al.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions. arXiv preprint arXiv:2311.05232 (2023)

[30] Ray, P.P.: Chatgpt: A comprehensive review on background, applications, key challenges, bias, ethics, limitations and future scope. Internet of Things and Cyber-Physical Systems (2023)

[31] Bhavya, B., Xiong, J., Zhai, C.: Analogy generation by prompting large language models: A case study of instructgpt. arXiv preprint arXiv:2210.04186 (2022)

[32] Chaparro, O., Florez, J.M., Marcus, A.: Using bug descriptions to reformulate queries during text-retrieval-based bug localization. Empirical Software Engineering **24**, 2947–3007 (2019)

[33] Mills, C., Parra, E., Pantiuchina, J., Bavota, G., Haiduc, S.: On the relationship between bug reports and queries for text retrieval-based bug localization. Empirical Software Engineering **25**, 3086–3127 (2020)

[34] Budhiraja, A., Dutta, K., Shrivastava, M., Reddy, R.: Towards word embeddings for improved duplicate bug report retrieval in software repositories. In: Proceedings of the 2018 ACM SIGIR International Conference on Theory of Information Retrieval. ICTIR '18, pp. 167–170. Association for Computing Machinery, New York, NY, USA (2018)

[35] Isotani, H., Washizaki, H., Fukazawa, Y., Nomoto, T., Ouji, S., Saito, S.: Duplicate bug report detection by using sentence embedding and fine-tuning. In: 2021 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 535–544 (2021). IEEE

[36] Lee, J., Han, K., Yu, H.: A light bug triage framework for applying large pre-trained language model. In: 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 1–11 (2022)

[37] Mani, S., Sankaran, A., Aralikatte, R.: Deeptriage: Exploring the effectiveness of deep learning for bug triaging. In: Proceedings of the ACM India Joint International Conference on Data Science and Management of Data, pp. 171–179 (2019)

[38] Zhou, Y., Sharma, A.: Automated identification of security issues from commit messages and bug reports. In: Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering. ESEC/FSE 2017, pp. 914–919. Association for Computing Machinery, New York, NY, USA (2017)

[39] Liu, H., Yu, Y., Li, S., Guo, Y., Wang, D., Mao, X.: Bugsum: Deep context understanding for bug report summarization. In: Proceedings of the 28th International

Conference on Program Comprehension, pp. 94–105 (2020)

[40] Dong, J., Lou, Y., Zhu, Q., Sun, Z., Li, Z., Zhang, W., Hao, D.: FIRA: fine-grained graph-based code change representation for automated commit message generation. In: 44th IEEE/ACM 44th International Conference on Software Engineering, ICSE 2022, Pittsburgh, PA, USA, May 25-27, 2022, pp. 970–981. ACM, New York, NY, USA (2022). https://doi.org/10.1145/3510003.3510069 . https://doi.org/10.1145/3510003.3510069

[41] Ahmed, T., Devanbu, P.: Few-shot training llms for project-specific code-summarization. In: 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 1–5 (2022)

[42] Rastkar, S., Murphy, G.C., Murray, G.: Automatic summarization of bug reports. IEEE Transactions on Software Engineering **40**(4), 366–380 (2014)

[43] Li, X., Jiang, H., Liu, D., Ren, Z., Li, G.: Unsupervised deep bug report summarization. In: Proceedings of the 26th Conference on Program Comprehension, pp. 144–155 (2018)

[44] Ma, X., Keung, J.W., Yu, X., Zou, H., Zhang, J., Li, Y.: Attsum: A deep attention-based summarization model for bug report title generation. IEEE Transactions on Reliability (2023)

[45] Zhang, T., Irsan, I.C., Thung, F., Han, D., Lo, D., Jiang, L.: Automatic pull request title generation. In: 2022 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 71–81 (2022). IEEE

[46] Liu, Z., Xia, X., Treude, C., Lo, D., Li, S.: Automatic generation of pull request descriptions. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 176–188 (2019). IEEE

[47] Jiang, S.: Boosting neural commit message generation with code semantic analysis. In: 2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 1280–1282 (2019). https://doi.org/10.1109/ASE.2019.00162

[48] Wang, H., Xia, X., Lo, D., Grundy, J., Wang, X.: Automatic solution summarization for crash bugs. In: 2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE), pp. 1286–1297 (2021). IEEE

[49] Xie, R., Hu, T., Ye, W., Zhang, S.: Low-resources project-specific code summarization. In: 37th IEEE/ACM International Conference on Automated Software Engineering, pp. 1–12 (2022)

[50] Jiang, H., Zhu, J., Yang, L., Liang, G., Zuo, C.: Deeprelease: Language-agnostic release notes generation from pull requests of open-source software. In: 2021

28th Asia-Pacific Software Engineering Conference (APSEC), pp. 101–110 (2021). IEEE