iTiger: An Automatic Issue Title Generation Tool

Ting Zhang*
tingzhang.2019@phdcs.smu.edu.sg
Singapore Management University
Singapore

Ivana Clairine Irsan*
ivanairsan@smu.edu.sg
Singapore Management University
Singapore

Ferdian Thung
ferdianthung@smu.edu.sg
Singapore Management University
Singapore

DongGyun Han
dhan@smu.edu.sg
Singapore Management University
Singapore

David Lo
davidlo@smu.edu.sg
Singapore Management University
Singapore

Lingxiao Jiang
lxjiang@smu.edu.sg
Singapore Management University
Singapore

ABSTRACT

In both commercial and open-source software, bug reports or issues are used to track bugs or feature requests. However, the quality of issues can differ a lot. Prior research has found that bug reports with good quality tend to gain more attention than the ones with poor quality. As an essential component of an issue, title quality is an important aspect of issue quality. Moreover, issues are usually presented in a list view, where only the issue title and some metadata are present. In this case, a concise and accurate title is crucial for readers to grasp the general concept of the issue and facilitate the issue triaging. Previous work formulated the issue title generation task as a one-sentence summarization task. A sequence-to-sequence model was employed to solve this task. However, it requires a large amount of domain-specific training data to attain good performance in issue title generation. Recently, pre-trained models, which learned knowledge from large-scale general corpora, have shown much success in software engineering tasks.

In this work, we make the first attempt to fine-tune BART, which has been pre-trained using English corpora, to generate issue titles. We implemented the fine-tuned BART as a web tool named iTiger, which can suggest an issue title based on the issue description. iTiger is fine-tuned on 267,094 GitHub issues. We compared iTiger with the state-of-the-art method, i.e., iTAPE, on 33,438 issues. The automatic evaluation shows that iTiger outperforms iTAPE by 29.7%, 50.8%, and 34.1%, in terms of ROUGE-1, ROUGE-2, ROUGE-L F1-scores. The manual evaluation also demonstrates the titles generated by BART are preferred by evaluators over the titles generated by iTAPE in 72.7% of cases. Besides, the evaluators deem our tool as useful and easy-to-use. They are also interested to use our tool in the future.

Demo URL: https://tinyurl.com/itiger-tool

Source code and replication package URL: https://github.com/soarsmu/iTiger

*Both authors contributed equally to this research.

ESEC/FSE 2022, 14 - 18 November, 2022, Singapore

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/XXXXXXX.XXXXXXX

CCS CONCEPTS

• **Software and its engineering → Software maintenance tools**.

KEYWORDS

issues, bug reports, title generation, pre-trained models

1 INTRODUCTION

In software development and maintenance, bug reports or issues[1] are heavily used by developers to report bugs or propose new features. With the proliferation of open-source software and social coding platforms, issue trackers is more accessible than ever. On one hand, given their varying experience levels, developers write bug reports with various qualities. On the other hand, the quality of bug reports can impact the bug triaging process. Prior research [5] finds that well-written bug reports are more likely to gain triager's attention and influence the decision on whether the bugs get fixed.

There is an emerging research interest in improving issue quality. An issue usually includes a title and a description. The description is optional and can contain extensive and rich information, such as detailed steps to reproduce a bug. The title serves as the summary of the description. Most prior works focus on improving the description of an issue and not its title [3]. For instance, Chaparro et al. [3] consider that a good bug report description should clearly describe the Observed Behavior (OB), the Steps to Reproduce (S2R), and the Expected Behavior (EB). They propose an approach to improve bug descriptions quality by alerting reporters about missing EB and S2R at reporting time. Compared to the attention given to the quality of issue description, the quality of issue titles has only recently received research interests. Succinct and accurate issue titles can help readers to quickly grasp the issue content and potentially speed up the bug triaging process, especially when issues are presented in the form of list. However, since developers may neglect to compose a succinct and accurate issue title, there is a need for an automatic issue title generation tool to help developers.

Chen et al. [4] are the first to work on the issue title generation task, which aims to help developers write issue titles. They formulated the issue title generation task as a one-sentence summarization task. They propose iTAPE [4], which is a specialized tool to generate the issue title based on the issue description. iTAPE relies on a sequence-to-sequence model [13] and is implemented with the OpenNMT framework[2]. Different from them, we leverage

[1]Note in our work, we use the terms *bug report* and *issue* interchangeably.
[2]https://github.com/OpenNMT/OpenNMT-py

pre-trained models (PTMs), which have brought considerable breakthroughs in the field of artificial intelligence. PTMs are pre-trained in a large amount of unlabeled data, and can be fine-tuned for solving downstream tasks. Fine-tuning PTMs can usually achieve better performance than learning models from scratch. To fill the gap of adopting PTMs to solve issue title generation task, in this work, we leverage a type of PTMs, i.e., BART [7], which has demonstrated promising performance in summarization and text generation tasks. Specifically, we benefit from transfer learning, where the BART model we used has been pre-trained in large English context. The model is further fine-tuned using a GitHub issue title dataset collected by Chen et al. [4], which consists of 333,563 issues.

To build a bridge between research and practice, we present iTiger, a web-based tool to generate high-quality issue titles. To use iTiger, developers simply need to install a Userscript manager, i.e., `Tampermonkey` [1], that is available in most popular web browsers, such as Chrome, Safari, and Firefox. After that, developers can install the script we provided. iTiger is specially designed for the scenario when developers are creating a new issue: they can focus on drafting a detailed description, and iTiger can automatically generate a succinct and accurate summary of the description. The script of iTiger is published on GitHub.[3]

2 APPROACH

We demonstrate the overall workflow of iTiger in Figure 1. First, when issue reporters create a new issue, they need to fill in the description field of the issue (as shown in Figure 2). Second, upon finishing writing the issue description, they can click the *Get Title Suggestion* button (as the red box shown in Figure 2). iTiger then sends a request to the backend to generate the title. After getting the title, iTiger will auto-fill the title field of the issue. Third, issue reporters can further modify and polish the issue title if they choose to do so. Fourth, after they are satisfied with the issue title, they can proceed to open this new issue.

The underlying model of iTiger is BART, a standard sequence-to-sequence (Seq2Seq) Transformer architecture which has been pre-trained by first corrupting a text with noising functions, and then learning to reconstruct the original text [7]. Several noising functions are applied by BART, such as token masking, token deletion, and sentence permutation (sentences are shuffled in random order). In the original paper, evaluation on two news summarization datasets indicate that BART outperforms all existing works. BART is pre-trained in the same corpora as RoBERTa, which includes over 160GB of uncompressed English text [10]. In this work, we adopt the base model of BART [4] with 6 layers in the encoder and decoder. BART was fine-tuned as a standard sequence-to-sequence model from the source sequence to the target sequence. Specifically, in our task, the source sequence is the issue description, and the target sequence is the issue title. We fine-tune BART by feeding it with the pairs of issue description and issue title. In the inference stage, the input is the issue description and BART can generate the issue title as the output. We leave all the hyper-parameters values to their defaults. The detailed hyper-parameter settings are available in our replication package. We trained BART on the issue description and title pairs extracted from GitHub. The details about the data we used can be found in Section 4. Once the model is fine-tuned, we can utilize the model to generate the suggested title based on the issue description.

[3]https://github.com/soarsmu/iTiger
[4]https://huggingface.co/facebook/bart-base

3 IMPLEMENTATION DETAILS

We have implemented iTiger as a Userscript which can be run in popular web browsers. We believe, by seamlessly integrating iTiger with GitHub web UI (as opposed to making iTiger a standalone application), context switches between a standalone application and a browser are eliminated and therefore it saves developers time. To generate issue titles more accurately, we first fine-tune the pre-trained BART on domain-specific issue dataset. We fine-tuned BART with 3 NVIDIA Tesla V100 GPUs. Using fine-tuned BART model [7], iTiger will generate a title suggestion based on the issue description. Instead of directly submitting a new issue with the generated title, iTiger only fill the title field so the reporter has a freedom to modify the title if deemed necessary.

iTiger's architecture consists of two components: frontend Userscript and backend. Frontend Userscript constructs the user interface. Backend provides a RESTful API for easy integration, such that a single deployment can serve multiple clients.

Frontend Userscript: The Userscript constructs iTiger's user interface and integrates it with the backend service. The Userscript adds a `Get Title Suggestion` button on the new issue page of GitHub. When the button is clicked, the Userscript sends a HTTP request to the backend service and fills the title field with suggested title from the backend service.

Backend: iTiger's backend provides a RESTful API to serve the clients. It is developed using Python 3, utilizing FastAPI framework. By leveraging server-client architecture, one backend service could serve multiple clients, and the clients do not need to store the model in their machine. iTiger's backend server passes the description to iTiger's title generator model every time there is an incoming request.

Deployment: iTiger's backend service is made available in our replication package. It can be deployed in any machine via docker containerization by following the step-by-step guide provided in our replication package. Note that one backend service can serve multiple clients. To be able to use iTiger, the client's side (i.e., issue reporter) needs to install `Tampermonkey` extension in their browser and import our Userscript on it. It will add a the `Get Title Suggestion` button. When this button is clicked, it will send a request and fill the title field in the GitHub's new issue page.

4 EVALUATION

4.1 Evaluation Setup

Dataset: We adopt the publicly available issue title generation dataset provided by Chen et al. [4]. This dataset contains 333,563 issues collected from the top-200 most-starred repositories on GitHub. These issues have high-quality titles. Three heuristic rules were adopted to select the issues: (1) issues whose titles have less than 5 words, or more than 15 words; (2) issues whose titles have more than 70% words missing in the description; (3) issues whose titles have a sub-sequence can exactly match a particular part of the issue description and this sub-sequence is over 70% of the title length are filtered out. More information about the dataset can be found in

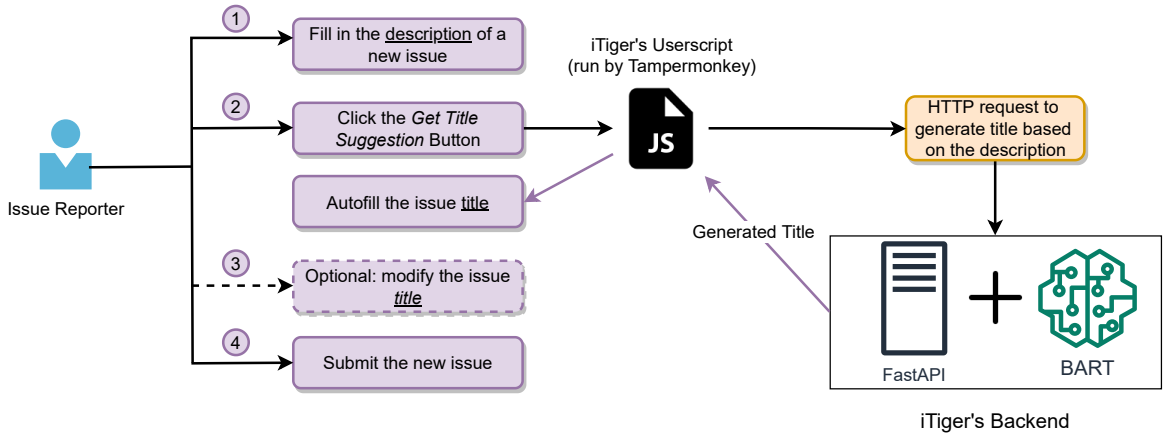

Figure 1: The scenario of using iTiger

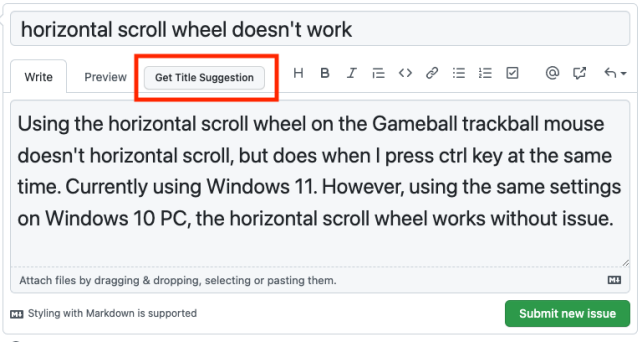

Figure 2: The title field is filled in by iTiger

the previous work [4]. The training, validation, and test data was split by the ratio of 8:1:1.

Automatic Evaluation: We apply ROUGE metric [8] to evaluate the accuracy of generated issue titles. Specifically, we report ROUGE-N (N=1,2) and ROUGE-L, which have been widely used in prior summarization papers [4, 11]. The recall, precision, and F1-score for ROUGE-N are calculated as follows:

$$Recall_{rouge-n} = \frac{count(overlapped\_N\_grams)}{count(N\_grams \in ref\_summary)}$$

$$Precision_{rouge-n} = \frac{count(overlapped\_N\_grams)}{count(N\_grams \in gen\_summary)}$$

$$F1_{rouge-n} = 2 \times \frac{Recall_{rouge-n} \times Precision_{rouge-n}}{Recall_{rouge-n} + Precision_{rouge-n}}$$

In the above equations, reference summary ($ref\_summary$) refers to the original issue title, while generated summary ($gen\_summary$) refers to the title generated by models. ROUGE-1 and ROUGE-2 differ in whether we count uni-gram or bi-grams. $overlapped\_N\_grams$ indicates the N-grams exist in both the reference summary and the generated summary. Thus, using the above equations, Recall measures the percentage of the N-grams in the reference summary that has been covered by the generated summary, while Precision measures the percentage of N-grams in the generated summary that really exist in the reference summary. F1-score considers both Precision and Recall. Thus, in our work, we treat F1-score as the evaluation metric. Slightly different from ROUGE-N, ROUGE-L F1-score is based on Longest Common Subsequence (LCS). It compares the similarity between two given texts in automatic summarization evaluation. We report ROUGE-N (N=1,2), ROUGE-L F1-score in our work and we simply refer them as ROUGE-1, ROUGE-2, and ROUGE-L. To understand whether iTiger can generate better titles than the state-of-the-art approach iTAPE can do, we report the results of the automatic evaluation on iTiger and iTAPE.

Manual Evaluation: Since the goal of iTiger is to help practical use, we conducted two types of manual evaluation: one focuses on comparing the accuracy between the title generated by BART and iTAPE, while the other focuses on the usability of the tool.

Accuracy: We randomly sampled 30 issues from the test set. We invited 5 evaluators: 3 Ph.D. students in Computer Science and 2 Research Engineers in Software Engineering. They have programming experience of more than 5 years and have been using GitHub for more than two years. We provided 30 issue descriptions with two generated titles (one was generated by iTiger, the other was generated by iTAPE). We also randomize the order of the titles produced by the two tools presented to the evaluators. The evaluators have no knowledge about the authorship of the two titles.

Usability: We asked the same 5 evaluators to install and use iTiger directly on GitHub. We asked them to provide three scores, each score ranges from 1 to 5 (strongly disagree, disagree, slightly agree, agree, and strongly agree). The three scores evaluate three aspects: iTiger is easy-to-use, iTiger is useful, and they would like to use iTiger in the future.

4.2 Result

Automatic Evaluation: Table 1 shows the ROUGE-1, ROUGE-2, and ROUGE-L F1-scores produced by the two models, including iTiger and iTAPE. Since we used the exact splits as iTAPE, we directly cite the results of iTAPE from its paper. We observe that iTiger outperforms iTAPE by 29.7%, 50.7%, and 34.1%, in terms of ROUGE-1 F1 score, ROUGE-2 F1 score, and ROUGE-L F1 score, respectively.

Table 1: Results on automatic evaluation

Approach	ROUGE-1	ROUGE-2	ROUGE-L
iTAPE	31.36	13.12	27.79
iTiger	40.67	20.6	37.26

Table 2: Results on manual evaluation

Approach	#Preferred
iTAPE	41
iTiger	109

Manual Evaluation: Table 2 shows the number of titles generated by the two tools that are preferred by evaluators. The titles generated by iTiger are preferred by evaluators on about 72.7% cases. In terms of usability, the evaluators consider iTiger to be easy-to-use (5 out of 5) and useful (3.8 out of 5) and are willing to use iTiger in the future (4.6 out of 5).

4.3 Threats to Validity

Threats to internal validity relate to the experimental bias. Following prior works on summarization studies [11, 14], we adopted both automatic evaluation (i.e., ROUGE metrics) and manual evaluation. Like other manual evaluation, our experimental results may be biased. To minimize the potential biases, we invited 5 evaluators from our research group. They have a programming experience of more than 5 years and have been using GitHub for more than two years. The authorship of the issue titles were hidden when they conducted the evaluation. *Threats to external validity* relate to whether our findings can be generalized to other datasets. To alleviate the external threats, we evaluate our approach on the dataset provided by Chen et al. [4], which consists of issues from top-200 most-starred repositories.

5 RELATED WORK

Issue Quality Understanding and Improvement. Existing research suggests that high-quality issues tend to get more attention than those with poor-quality, and easier-to-read bug reports have shorter lifetimes [2]. Bettenburg et al. [2] conduct a survey among developers to investigate the quality of bug reports from developers' perspective. They consider several fine-grained information in the issue description, such as code samples and stack traces. Their findings include but not limited to the fact that bug reports containing stack traces get fixed sooner and easy-to-read reports are fixed faster. Guo et al. [5] also confirm the finding that high-quality bug reports are more likely to gain the triager's attention, especially if they contain clear steps for reproducing the bug.

Although our task focuses only on a part of issues, i.e., generating issue title generation, our final goal is to help improve issue quality. Our strategy is to suggest a high-quality issue title instead of detecting existing bad ones. Our work complements the existing works that aim to improve bug report quality.

Software Artifact Generation. Our work on issue title generation belongs to a broader research topic of software artifact generation. There have been several tools proposed to automatically generate different software artifacts, such as bug reports [9, 12], pull request titles [14], and pull request descriptions [11].

Bug report summarization has attracted research interests for more than one decade [12]. Although it shares similarities with our task, there are some important differences. An obvious difference is the length of the target sequence: bug report summaries usually contain several sentences, whereas an issue title is a single-sentence summary. Recently, Liu et al. [9] propose an unsupervised approach that first converts sentences to vectors and then leverages an auto-encoder network to extract semantic features. They also utilize interactive discussions, i.e., comments of a bug report, to measure whether a sentence is approved or disapproved. Recently, an issue title generation tool, i.e., iTAPE, was proposed by Chen et al. [4]. Chen el at. [4] collected issues from GitHub and created the first benchmark. Despite of relatively good performance achieved by iTAPE, there is still a room for improvement.

Furthermore, we recently worked on the task of pull request title generation [14]. We evaluate several state-of-the-art summarization approaches in the dataset we built. Both the automatic and manual evaluation results indicate that the fine-tuned `BART-base` model achieved the best performance in the pull request title generation task. We also provided an associated tool named AutoPRTitle [6], which aims to automatically generate pull request titles. Inspired by our earlier work [14], we apply pre-trained BART to the issue title generation task. iTiger differs from AutoPRTitle in several aspects: (1) the data source: Pull request title generation requires more data sources, including a pull request description, commit messages, and the associated issue titles. In comparison, issue title generation only requires an issue description. (2) the tool usage: We implement these two tools considering the different usages between them. Since issue generation only requires an issue description, we implemented iTiger as a Userscript that embeds a user interface directly on the GitHub's issue creation page to reduce the context-switching between applications. On the other hand, pull request title generation require more data sources. Hence, we implemented AutoPRTitle as a stand-alone web application to better capture these different sources of information.

6 CONCLUSION AND FUTURE WORK

In this paper, we present iTiger, which generates the issue title based on the issue description. iTiger allows developers to modify the generated titles. iTiger utilizes the state-of-the-art summarization model, i.e., BART. In automatic evaluation, iTiger outperforms the prior specialized issue title generation model, i.e., iTAPE. In manual evaluation, the evaluators indicate that they prefer the titles generated by BART over the ones generated by iTAPE in 73.1% cases. They agree that iTiger is easy to use and useful, and they are also willing to use our tool in their daily work. In the future, we would like to train iTiger on larger and more data to improve the quality of generated issue titles.

Acknowledgment This research / project is supported by the National Research Foundation, Singapore, under its Industry Alignment Fund – Pre-positioning (IAF-PP) Funding Initiative. Any opinions, findings and conclusions or recommendations expressed in this material are those of the author(s) and do not reflect the views of National Research Foundation, Singapore.

REFERENCES

[1] [n.d.]. Tampermonkey Home Page. https://www.tampermonkey.net/. (Accessed on 05/21/2022).

[2] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of software engineering*. 308–318.

[3] Oscar Chaparro, Jing Lu, Fiorella Zampetti, Laura Moreno, Massimiliano Di Penta, Andrian Marcus, Gabriele Bavota, and Vincent Ng. 2017. Detecting missing information in bug descriptions. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*. 396–407.

[4] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2020. Stay professional and efficient: Automatically generate titles for your bug reports. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 385–397.

[5] Philip J Guo, Thomas Zimmermann, Nachiappan Nagappan, and Brendan Murphy. 2010. Characterizing and predicting which bugs get fixed: an empirical study of microsoft windows. In *Proceedings of the 32Nd ACM/IEEE International Conference on Software Engineering-Volume 1*. 495–504.

[6] Ivana Clairine Irsan, Ting Zhang, Ferdian Thung, David Lo, and Lingxiao Jiang. 2022. AutoPRTitle: A Tool for Automatic Pull Request Title Generation. In *2022 IEEE 38th International Conference on Software Maintenance and Evolution (ICSME)*. Tool Demo Track.

[7] Mike Lewis, Yinhan Liu, Naman Goyal, Marjan Ghazvininejad, Abdelrahman Mohamed, Omer Levy, Veselin Stoyanov, and Luke Zettlemoyer. 2020. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*. Association for Computational Linguistics, 7871–7880.

[8] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.

[9] Haoran Liu, Yue Yu, Shanshan Li, Yong Guo, Deze Wang, and Xiaoguang Mao. 2020. Bugsum: Deep context understanding for bug report summarization. In *Proceedings of the 28th International Conference on Program Comprehension*. 94–105.

[10] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[11] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2019. Automatic generation of pull request descriptions. In *2019 34th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 176–188.

[12] Sarah Rastkar, Gail C Murphy, and Gabriel Murray. 2010. Summarizing software artifacts: a case study of bug reports. In *2010 ACM/IEEE 32nd International Conference on Software Engineering*, Vol. 1. IEEE, 505–514.

[13] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. *Advances in neural information processing systems* 27 (2014).

[14] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, Donggyun Han, David Lo, and Lingxiao Jiang. 2022. Automatic Pull Request Title Generation. In *2022 IEEE 38th International Conference on Software Maintenance and Evolution (ICSME)*. Research Track.