Too long; didn't read: Automatic summarization of GitHub README.MD with Transformers

Thu T. H. Doan
doanthihoaithu@vnu.edu.vn
VNU University of Engineering & Technology
Hanoi, Vietnam

Phuong T. Nguyen
phuong.nguyen@univaq.it
University of L'Aquila
L'Aquila, Italy

Juri Di Rocco
juri.dirocco@univaq.it
University of L'Aquila
L'Aquila, Italy

Davide Di Ruscio
davide.diruscio@univaq.it
University of L'Aquila
L'Aquila, Italy

ABSTRACT

The ability to allow developers to share their source code and collaborate on software projects has made GitHub a widely used open source platform. Each repository in GitHub is generally equipped with a README.MD file to exhibit an overview of the main functionalities. Nevertheless, while offering useful information, README.MD is usually lengthy, requiring time and effort to read and comprehend. Thus, besides README.MD, GitHub also allows its users to add a short description called "About," giving a brief but informative summary about the repository. This enables visitors to quickly grasp the main content and decide whether to continue reading. Unfortunately, due to various reasons–not excluding laziness–oftentimes this field is left blank by developers.

This paper proposes GitSum as a novel approach to the summarization of README.MD. GitSum is built on top of BART and T5, two cutting-edge deep learning techniques, learning from existing data to perform recommendations for repositories with a missing description. We test its performance using two datasets collected from GitHub. The evaluation shows that GitSum can generate relevant predictions, outperforming a well-established baseline.

KEYWORDS

GitHub, README.MD, mining software repositories, recommender systems, summarization

ACM Reference Format:
Thu T. H. Doan, Phuong T. Nguyen, Juri Di Rocco, and Davide Di Ruscio. 2023. Too long; didn't read: Automatic summarization of GitHub README.MD with Transformers. In *Proceedings of the International Conference on Evaluation and Assessment in Software Engineering (EASE '23), June 14–16, 2023, Oulu, Finland*. ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3593434.3593448

EASE '23, June 14–16, 2023, Oulu, Finland

ACM ISBN 979-8-4007-0044-6/23/06…$15.00
https://doi.org/10.1145/3593434.3593448

1 INTRODUCTION

GitHub is the largest open source software store, where the majority of projects are publicly shared with the world, free of charge [4]. Each GitHub repository has one or more README.MD files to provide basic information about the repository, e.g., instructions, help, or details about updates, to name a few. Such a file is presented in the Markdown format to display information in an intuitive way. A well-written README.MD is an important component, allowing GitHub visitors to grasp the scope of a repository. In fact, README.MD is usually the first item that visitors consider when it comes to becoming acquainted with a repository [18].

By large-scale projects, where there are several software artifacts and components, the amount of information that a README.MD file needs to convey is large, and thus making it become very lengthy, requiring much efforts to read and understand. Altogether, this may discourage visitors from continuing with the respository. Therefore, apart from README.MD, GitHub also allows its users to add a short description called "About" to each repository, offering a succinct summary of the main functionalities. Fig. 1 shows an example with the DeepSpeed repository, seen from the Microsoft's GitHub frontpage.[1] Apart from information related to the number of forks, stars, there is also a short description, which is actually the "About" field, describing the main functionalities. Thanks to this, developers–especially first-time visitors–could gain a quick orientation to browse the source code, without looking into the corresponding README.MD, which resides behind the frontpage.

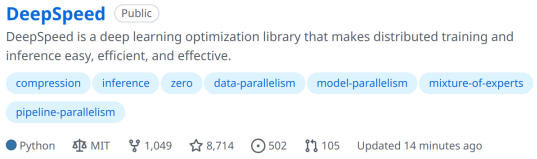
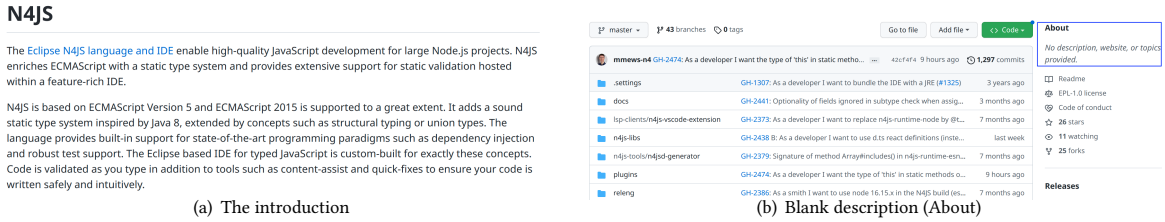


Figure 1: The DeepSpeed repository.

Nonetheless, in reality the "About" field is usually left unfilled by developers, possibly due to laziness. Fig 2 shows a repository as our motivating example. This repository contains the source code of the N4JS project[2] which is a general-purpose programming language based on ECMAScript, supporting static type systems for

[1]DeepSpeed is a project developed by Microsoft https://github.com/microsoft/DeepSpeed
[2]https://github.com/eclipse/n4js



(a) The introduction (b) Blank description (About)

Figure 2: The N4JS repository.

JavaScript applications. More specifically, Fig 2(a) shows part of its README.MD, and Fig 2(b) visualizes its structure in directory format. With around 1.8 millions lines of code, N4JS is a large project, resulting in a big README.MD to compose all effective information. However, as seen in Fig 2(b), no description is provided in the "About" section (the blue frame). The example suggests the need to automatically generate a short "About" description for a repository. Prior work has focused on summarization for other artifacts such as code [1, 27], issues [30], bug reports [5], pull requests [10, 19], but none of them has tried to generate "About" descriptions.

We propose GitSum, a workable solution to the summarization of README.MD files for GitHub repositories. GitSum is built on top of BART [14] and T5 [26], two pre-trained models that have shown superiority in several summarization tasks [28]. *To the best of our knowledge, GitSum is the first approach to offer this kind of recommendations.* Thus, our contributions are summarized below:

- We introduce GitSum as a novel approach to the summarization of README.MD, providing a brief but informative description of the repository under consideration.
- We perform an empirical evaluation on two datasets consisting of GitHub open-source projects.
- The datasets collected and the tool developed through this paper are made available to facilitate future research.[3]

2 RELATED WORK

This section reviews related studies which address the emerging summarization problems, including those in software engineering, i.e., generation of release notes, issue titles, and post titles.

2.1 Text summarization

Summarization has been conceived as an effective means to generate a shorter but informative summary from long pieces of text [14, 26]. There are two main types of approaches in automatic text summarization, namely extractive summarization and abstractive summarization. The former involves selecting important keyphrases from the original text and combining them to produce a summary without generating any new text. With various algorithms being proposed, the main tasks have been effectively resolved, including intermediate representation, sentence score, and summary sentence selection [24]. Meanwhile, the latter entails more deep understanding of the document, which could offer a realistic summary by employing paraphrasing sentences or new text generation [3, 24].

Extractive summarization has become a trendy research topic with a high level of maturity [9]. Starting with initial ideas based on scoring sentences [6, 25], various approaches with the help of neural networks recently have been proposed that show impressive results [2, 21]. Now the community has shifted towards abstractive summarization which requires deeper analysis of the input text in order to generate a more human-like summary [9, 11, 13, 20, 29].

In this work, we formulate the generation of descriptions from README.MD as an abstractive summarization task. We aim to produce realistic and useful summaries for GitHub repositories.

2.2 Summarization in Software Engineering

Recently, the SE community has deployed summarization in different tasks. Chen et al. [5] proposed iTAPE, an approach to the generation of high-quality titles for issues on GitHub, exploiting one-sentence summarization. Hao et al. [16] developed a quality prediction-based filter TitleGen-FL including the deep learning-based (DL) module and the information retrieval-based (IR) module. This filter can predict whether the method iTAPE can generate a high-quality title by analyzing the issue body.

Moreno et al. developed ARENA [22, 23], an approach for the automatic generation of release notes that combines in a unique way source code analysis, code summarization techniques, and information from versioning systems and issue trackers. The obtained results indicate that ARENA is effective in producing helpful release notes, as considered by professional developers. With a different direction, a recent study [12] introduced an approach named DeepRelease to generate release notes according to pull requests. The approach exploits useful information from pull requests, and then summarizes change entries in release notes effectively.

Studies have been conducted to generate titles for Stack Overflow posts. Gao et al. [8] were the first to automatically generate a post title according to a specific code snippet, implementing a tool named Code2Que which employs a sequence-to-sequence model as their backbone model. Liu et al. [17] considered both modalities (code snippet and problem description) in their work. They proposed SOTitle, which is fine-tuned based on the pre-trained transformer language model T5. The experimental results showed that SOTitle outperforms various baselines including Code2Que.

README.MD contains important information about source code, and it could scale dramatically with large-scale projects. As no studies have tackled the issue of README.MD summarization, we conduct this work with the aim of helping developers quickly catch up with a GitHub repository by means of a succinct summary.

3 PROPOSED APPROACH

The workflow of our GitSum is presented in Fig 3, including three main phases, i.e., *Data Collecting*, *Summarizing*, and *Generating &*

[3] https://github.com/MDEGroup/GitSum/

Evaluating. Firstly, README.MD data is fetched from real open-source projects in GitHub by means of the *Data Collecting* phase. After that, the data is pre-processed and fed to the selected pre-trained model for further training in the *Summarizing* phase. Finally, summaries for elements in test dataset are generated, and evaluated.

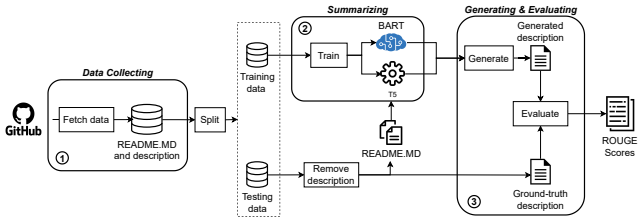


Figure 3: The GitSum workflow.

3.1 Data collecting

The extraction of data from GitHub repositories can be done in different ways, including GitHub API, GitHub snapshots, and GitHub GraphQL. We opted for GitHub GraphQL API to crawl repositories to provide input for the evaluation, as it offers a more flexible, efficient, and robust way of querying data compared to the other alternatives. GraphQL API allows for a strongly typed schema that defines the available data and the relationships between the data. A search query can be executed to perform a repository search on GitHub using GraphQL. The search facility requires three parameters, including a *query* to search for, the *type* of entity to search for, and the *number* of requested results (max 100).

Listing 1: Search parameters.

```
query:"topic:'<featured-topic>' stars:>500'# Introduction
    ' in:readme", type: Repository, first:100
```

As shown in Listing 1, we considered repositories with more than 500 stars to avoid toy or dummy ones. Since the GitHub Search API provides up to 1,000 search items, we used a python script to perform searches iteratively by varying the topics query element. Those topics are extracted for GitHub featured-topic list.[4] The script iterates among the result pages if needed. Because we are interested in summarizing README.MD files, we restrict the results to repositories that include structured README.MD files. The response to this query contains information about the repositories that match the search criteria, such as their name, description, number of stargazers, and the login of the repository owner.

3.2 Summarizing

▷ **BART.** This is a standard sequence-to-sequence Transformer architecture developed by Lewis et al. [14]. Built on top an encoder-decoder architecture, BART has demonstrated its ability to achieve state-of-the-art (SOTA) results in several summarization tasks. Furthermore, it was fine-tuned as standard Seq2Seq models in order to provide a quick solution for various sequence-to-sequence learning problems. It can be considered as generalizing BERT (due to the bidirectional encoder), GPT (with the left-to-right decoder), and many other more recent pretraining schemes. But following GPT, GELU is used as the activate function in BART instead of ReLU.

[4] https://github.com/topics

▷ **T5.** Text-to-Text-Transfer-Transformer (T5) was firstly introduced by Raffel et al. [26], and it becomes one of the state-of-the-art pre-trained models, which has shown to obtain a promising performance in solving NLP tasks. T5 is based on a Transformer architecture using a text-to-text approach as its input and output are always presented in text strings, and trained in a high quality pre-processed English corpus with 700GB in size [26]. T5 is pre-trained with the same objective to BERT [7], following the concepts of Masked Language Models. Masked Language Models are Bidirectional models in which the representation of a word is derived from both the left and the right of its context.

Table 1: Details of the base BART and T5.

Pre-trained model	BART	T5
Development team	Facebook	Google
# encoder-decoder layers	6	12
Embedding size	1,024	512
Beam size	4	4
Length penalty	1.0	1.0
# parameters	140M	220M

In this implementation, we adopted T5 and the base version of BART as the classification engine. We set all the hyper-parameters values as the default configuration. Details about the BART and T5 recommendation engines are depicted in Table 1. We continuously trained on README.MD files and their descriptions. For every README.MD and its description in the training dataset that could be considered as a pair of source sequence and target sequence, they are tokenized into token vectors by using the tokenizier going along with each pre-trained model. Afterward, the achieved token vectors of the source sequence and the target sequence are fed to the models to update weights.[5]

3.3 Generating & Evaluating

In this phase, the model is used to the prediction engine, which has been trained with several training repositories. README.MD files of testing repositories are used as input to generate summaries. The real label of each testing instance is removed to use as ground-truth data. The generated summaries are then compared with the real ones in the ground-truth data to evaluate the performance. In reality, GitSum should be trained with a lot of repositories to provide predictions for those without any "About" description.

4 EVALUATION

This section describes the methods and materials used to conduct an empirical evaluation on GitSum.

4.1 Research questions

We consider the following research questions:

▷ **RQ$_1$**: *Which technique between BART and T5 brings a better performance for GitSum?* BART [14] and T5 [26] have been conceived to deal with translation tasks. We evaluate which of them is more suitable for generating "About" summary from README.MD.

▷ **RQ$_2$**: *Compared to iTAPE, how effective is GitSum in summarizing README.MD?* To the best of our knowledge, no work has been conducted to tackle the issue of summarizing README.MD to yield

[5] The fine-tuning script and other materials related to our work are provided in the replication package.

Table 2: Examples of recommendation provided by GitSum.

No.	Ground-truth Description	Generated Description	ROUGE-1	ROUGE-2	ROUGE-L
1	compiled list of more than 350 resources to delve into the endless realm of blockchain technology and web3	A compiled list of more than 350 resources to delve into the endless realm of blockchain technology	0.914	0.909	0.914
2	Pilot project of Spring Boot with Kafka Streams for SMS Delivery Filtering	A pilot repo for Spring Boot with Kafka Streams for SMS Delivery Filtering	0.800	0.696	0.800
3	JSON query and transformation language	A complete query and transformation language for JSON	0.769	0.545	0.615
4	Listen to your to PostgreSQL database in realtime via websockets. Built with Elixir.	A server built with Elixir using the Phoenix Framework to listen to changes in your PostgreSQL database via logical replication and then broadcast those changes via WebSockets.	0.600	0.263	0.350
5	A light-weight monitoring tool with UI for user defined services and protocols	An extensible monitoring software tool that offers a light-weight User Interface to monitor the services and protocols defined by users	0.484	0.129	0.363

"About" summary. Thus, we adopted iTAPE [5], an approach for bug report summarization as the baseline to compare with GitSum.

4.2 Evaluation metrics

The ROUGE metrics [15] including ROUGE-1, ROUGE-2, ROUGE-L, are used to evaluate GitSum, as they have been widely used for text summarization tasks [5, 30], computed as follows:

$$Precision_{rouge-n} = \frac{\sum_{R,G \in S} \sum_{gram_n \in R} Count_G(gram_n)}{\sum_{R,G \in S} \sum_{gram_n \in G} Count_G(gram_n)}$$

$$Recall_{rouge-n} = \frac{\sum_{R,G \in S} \sum_{gram_n \in R} Count_G(gram_n)}{\sum_{R,G \in S} \sum_{gram_n \in R} Count_R(gram_n)}$$

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

where R, G, and S are the reference summary, generated summary, and the test set, respectively. $gram_n$ is an n-gram phase, where n presents for the length of a word sequence; $Count_R(gram_n)$ and $Count_G(gram_n)$ are the occurrence number of $gram_n$ in R and G.

ROUGE-1 and ROUGE-2 are computed using $N = 1, 2$, i.e., unigram and bi-grams. $\sum_{gram_n \in R} Count_G(gram_n)$ is the number of N-grams existing in both the reference summary and generated summary. The above equations can be interpreted as follows. Given two sets of N-grams produced from a generated summary and its original summary, respectively, Precision measures the proportion of N-grams in the first set that exists in the second set, while Recall gauges the proportion of N-grams in the second set that have been captured in the first set. Slightly different from ROUGE-N, ROUGE-L F1-score is based on Longest Common Subsequence (LCS):

$$Precision_{rouge-l} = \frac{LCS(R,G)}{length(G)}; Recall_{rouge-l} = \frac{LCS(R,G)}{length(R)}$$

$$F1_{rouge-l} = \frac{2 * Precision_{rouge-l} * Recall_{rouge-l}}{Precision_{rouge-l} + Recall_{rouge-l}}$$

where $LCS(R, G)$ is the length of the longest common subsequence of R and G, presenting for a reference summary and its generated summary, respectively. ROUGE-L indicates the natural similarity between the two given sequences in sentence-level structure. It is also considered to be useful in evaluating summarization performance.

4.3 Baseline and datasets

By investigating the existing literature, we realized that there has been no work with the same purpose, i.e., summarization of a README.MD to yield an "About" description. Thus, no approach can be considered as a direct baseline for our work. We looked for similar applications in Software Engineering, and came across iTAPE [5], a system for summarization of bug reports. To make a fair comparison with our proposed approach that focuses on a different domain, we only reused the given script for fine-tuning, skipping its pre-processing mechanism.

Two datasets consisting of README.MD files with their corresponding descriptions from GitHub repositories have been curated following the process described in Section 3.1. The first dataset contains 1,824 repositories which have README.MD file presented in English. The second dataset is curated by enriching the first one with data from GitHub by crawling repositories that match the search criteria, including their name, description, number of stargazers, and the login of the repository owner. Eventually, we obtained an additional dataset with 4,200 repositories. Following existing studies, we randomly split the dataset in the ratio of 80:20 for training and testing purposes. The splitting was done only once for the whole dataset, and we shuffled the data so as to randomly distribute the repositories. The final evaluation metrics are computed by averaging out the results obtained by all the testing instances.

5 RESULT ANALYSIS

Section 5.1 gives some recommendation examples provided by GitSum. Section 5.2 and Section 5.3 report and analyze the experimental results to answer the two research questions.

5.1 Recommendation examples

Table 2 lists some recommendation examples. The first column depicts the real descriptions, collected from GitHub. Meanwhile, the descriptions generated by GitSum are shown in the second column. The metrics for each pair of ground-truth data and recommendation are also reported.

Examples 1, 2, 3 show that GitSum can generate descriptions with highly close meaning to the real ones. Moreover, despite having lower ROUGE scores than the former, the last two samples demonstrate that the summary generated by GitSum brings even more useful information about the repository under consideration than the ground-truth specified by real users. We can see that by the examples, GitSum is able to provide meaningful recommendations.

5.2 RQ_1: *Which technique between BART and T5 brings a better performance for GitSum?*

We compare between the two GitSum configurations, i.e., BART and T5, to study which of them yields a superior summarization. Aiming for efficiency, we ran the two techniques using the small

dataset with 1,824 GitHub repositories. By an empirical evaluation, we noticed that when incorporating various pre-processing steps, e.g., stop-word removal, or lemmatization in our proposed solution, we did not get any significant improvement in the experimental results. Thus, we removed these steps from the pipeline.

Table 3: Performance of the two GitSum configurations.

Metric	BART	T5
Average ROUGE-1	**0.500**	0.484
Average ROUGE-2	**0.373**	0.363
Average ROUGE-L	**0.480**	0.461
Inference time	**1m43s**	3m28s

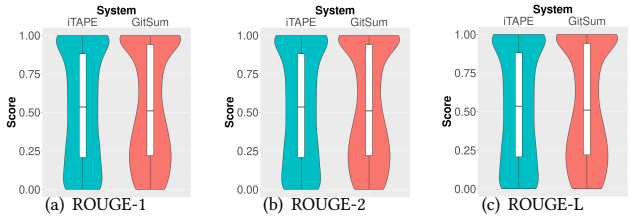
After fine-tuning with the training dataset, both BART and T5 are used to generate summary for README.MD in the test dataset, and then ROUGE scores are calculated for evaluation. Table 3 shows the experimental results achieved by the two configurations. The table demonstrates that BART obtains a better performance compared to T5 by all the considered metrics. In particular, BART yields 0.500, 0.373, and 0.480 as ROUGE-1, ROUGE-2, and ROUGE-L F1-score, respectively. Meanwhile, the corresponding scores by T5 are: 0.484, 0.363, and 0.461. Moreover, BART is also more timing efficient, i.e., it provides the prediction for all the testing instances within 1m43s, much faster in comparison with T5, which spends 3m48s for the same task. This essentially means that using BART to generate summarization for GitHub repositories brings both effectiveness and efficiency with respect to using T5. Nevertheless, the gain in accuracy obtained by BART compared to T5 is rather marginal.

Answer to RQ$_1$. *On the considered dataset, BART yields a slightly better summarization performance compared to T5. Moreover, it is also more efficient, generating recommendations in shorter time.*

5.3 RQ$_2$: *Compared to iTAPE, how effective is GitSum in summarizing README.MD?*

To further validate the generalizability of GitSum, we use the second dataset, which is larger than the first one in RQ$_1$, containing 4,200 repositories. By following the given descriptions of iTAPE in the original paper [5], we also fine-tune it on our training dataset and perform summary generation on the testing dataset.

The ROUGE scores achieved by iTAPE and GitSum are shown using the violin boxplots in Fig. 4. Such diagrams bring more information about the distribution of testing samples according to the ROUGE scores. We can see that the distributions of ROUGE metrics achieved by GitSum are presented in a shape that is slimmer around the middle and significantly wider at the top, compared to iTAPE. In other words, the amplitude of scores for GitSum in the top level, i.e., 1.0, is larger than that of iTAPE, i.e., the scores achieved by our approach are higher concentrated around the upper quartile, implying that GitSum provides effective summaries for more README.MD samples than iTAPE. This is further confirmed when we look into the whiskers of GitSum in the middle of the figures, i.e., they are in a higher level compared to those of iTAPE. By computing the metrics pairwise, we got the following improvement obtained by GitSum with respect to iTAPE: 3.85%, 1.35%, and 2.62% in terms of average ROUGE-1, ROUGE-2, and ROUGE-L scores, respectively.



(a) ROUGE-1 (b) ROUGE-2 (c) ROUGE-L

Figure 4: RQ$_2$: The experimental results.

Answer to RQ$_2$. *GitSum is more effective than iTAPE, as it provides more closely relevant descriptions with respect to all the three metrics, i.e., ROUGE-1, ROUGE-2, and ROUGE-L.*

Discussion. While GitSum yields a better performance compared to iTAPE in all the metrics, such a gain is considerably small, and thus we still see the room for improvement. We anticipate that additional pre-processing steps such as data cleaning and alignment could help GitSum further improve its performance. Due to space limits, we are not able to address this issue in this paper, and thus we consider as our future work. There are the following remarks: *(i)* it is necessary to anticipate the limitation of the summarization the risks with GitSum; and *(ii)* according to the results in Table 2, we can see that even when the ROUGE scores are considerably low, the generated descriptions are more informative and meaningful compared to the ground-truth one. Essentially, this triggers the need to propose more suitable metrics for measuring the relevance/serendipity between generated and real text.

5.4 Threats to validity

▷ **Internal validity.** This concerns how well our evaluation resembles real-world scenarios. In the evaluation, we crawled data from GitHub, and for the testing projects, we removed their real descriptions, and saved as ground-truth data. This simulates a real development scenario where we need to recommend summary for repositories using their README.MD.

▷ **External validity.** This is related to the generalizability of the findings outside this study. We attempted to mitigate the threats by evaluating GitSum using different experimental configurations to simulate real-world scenarios. Since there exists no approach with the same purpose, iTAPE–a tool for generation of titles for bug reports–was chosen as the baseline.

▷ **Construct validity.** This threat is related to the experimental settings to compare GitSum with iTAPE. To aim for a reliable comparison, we used the original implementation of iTAPE made available by its authors,[6] keeping the internal design unchanged. Moreover, we tried to employ the same settings to experiment with both systems.

6 CONCLUSION AND FUTURE WORK

This paper presented GitSum as a workable solution to the summarization of README.MD to yield the "About" summary for a repository. BART and T5 were used as the classification engine,

[6] https://github.com/imcsq/iTAPE

and the experimental results showed that GitSum obtains an encouraging recommendation performance, outperforming a baseline. For future work, we plan to evaluate GitSum with more data collected from GitHub. Moreover, we will try to improve the overall performance by incorporating different pre-processing steps, and summarization techniques, apart from BART and T5.

ACKNOWLEDGMENTS

This work has been partially supported by the EMELIOT national research project, which has been funded by the MUR under the PRIN 2020 program (Contract 2020W3A5FY).

REFERENCES

[1] Wasi Uddin Ahmad, Saikat Chakraborty, Baishakhi Ray, and Kai-Wei Chang. 2020. A Transformer-based Approach for Source Code Summarization. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault (Eds.). Association for Computational Linguistics, 4998–5007. https://doi.org/10.18653/v1/2020.acl-main.449

[2] Deepa Anand and Rupali Wagh. 2022. Effective deep learning approaches for summarization of legal texts. *Journal of King Saud University - Computer and Information Sciences* 34, 5 (2022), 2141–2150. https://doi.org/10.1016/j.jksuci.2019.11.015

[3] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2014. Neural Machine Translation by Jointly Learning to Align and Translate. https://doi.org/10.48550/ARXIV.1409.0473

[4] Hudson Borges and Marco Tulio Valente. 2018. What's in a GitHub Star? Understanding Repository Starring Practices in a Social Coding Platform. *Journal of Systems and Software* 146 (2018), 112–129. https://doi.org/10.1016/j.jss.2018.09.016

[5] Songqiang Chen, Xiaoyuan Xie, Bangguo Yin, Yuanxiang Ji, Lin Chen, and Baowen Xu. 2021. Stay Professional and Efficient: Automatically Generate Titles for Your Bug Reports. In *Proceedings of the 35th IEEE/ACM International Conference on Automated Software Engineering* (Virtual Event, Australia) *(ASE '20)*. Association for Computing Machinery, New York, NY, USA, 385–397. https://doi.org/10.1145/3324884.3416538

[6] Hans Christian, Mikhael Agus, and Derwin Suhartono. 2016. Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-IDF). *ComTech: Computer, Mathematics and Engineering Applications* 7 (12 2016), 285. https://doi.org/10.21512/comtech.v7i4.3746

[7] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2018. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. https://doi.org/10.48550/ARXIV.1810.04805

[8] Zhipeng Gao, Xin Xia, John Grundy, David Lo, and Yuan-Fang Li. 2020. Generating Question Titles for Stack Overflow from Mined Code Snippets. *ACM Trans. Softw. Eng. Methodol.* 29, 4, Article 26 (sep 2020), 37 pages. https://doi.org/10.1145/3401026

[9] Som Gupta and S. K Gupta. 2019. Abstractive summarization: An overview of the state of the art. *Expert Systems with Applications* 121 (2019), 49–65. https://doi.org/10.1016/j.eswa.2018.12.011

[10] 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 International Conference on Software Maintenance and Evolution (ICSME)*. 454–458. https://doi.org/10.1109/ICSME55016.2022.00058

[11] Yuuki Iwasaki, Akihiro Yamashita, Yoko Konno, and Katsushi Matsubayashi. 2019. Japanese abstractive text summarization using BERT. In *2019 International Conference on Technologies and Applications of Artificial Intelligence (TAAI)*. 1–5. https://doi.org/10.1109/TAAI48200.2019.8959920

[12] Huaxi Jiang, Jie Zhu, Li Yang, Geng Liang, and Chun Zuo. 2021. DeepRelease: Language-agnostic Release Notes Generation from Pull Requests of Open-source Software. In *2021 28th Asia-Pacific Software Engineering Conference (APSEC)*. 101–110. https://doi.org/10.1109/APSEC53868.2021.00018

[13] Hyunsoo Lee, YunSeok Choi, and Jee-Hyong Lee. 2020. Attention History-Based Attention for Abstractive Text Summarization. In *Proceedings of the 35th Annual ACM Symposium on Applied Computing* (Brno, Czech Republic) *(SAC '20)*. Association for Computing Machinery, New York, NY, USA, 1075–1081. https://doi.org/10.1145/3341105.3373892

[14] 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. 7871–7880. https://doi.org/10.18653/v1/2020.acl-main.703

[15] Chin-Yew Lin. 2004. ROUGE: A Package for Automatic Evaluation of Summaries. In *Text Summarization Branches Out*. Association for Computational Linguistics, Barcelona, Spain, 74–81. https://aclanthology.org/W04-1013

[16] Hao Lin, Xiang Chen, Xuejiao Chen, Zhanqi Cui, Yun Miao, Shan Zhou, Jianmin Wang, and Zhan Su. 2023. TitleGen-FL: Quality prediction-based filter for automated issue title generation. *Journal of Systems and Software* 195 (2023), 111513. https://doi.org/10.1016/j.jss.2022.111513

[17] Ke Liu, Guang Yang, Xiang Chen, and Chi Yu. 2022. SOTitle: A Transformer-based Post Title Generation Approach for Stack Overflow. https://doi.org/10.48550/ARXIV.2202.09789

[18] Yuyang Liu, Ehsan Noei, and Kelly Lyons. 2022. How ReadMe files are structured in open source Java projects. *Information and Software Technology* 148 (2022), 106924. https://doi.org/10.1016/j.infsof.2022.106924

[19] Zhongxin Liu, Xin Xia, Christoph Treude, David Lo, and Shanping Li. 2020. Automatic Generation of Pull Request Descriptions. In *Proceedings of the 34th IEEE/ACM International Conference on Automated Software Engineering* (San Diego, California) *(ASE '19)*. IEEE Press, 176–188. https://doi.org/10.1109/ASE.2019.00026

[20] M. Jishma Mohan, C. Sunitha, Amal Ganesh, and A. Jaya. 2016. A Study on Ontology Based Abstractive Summarization. *Procedia Computer Science* 87 (2016), 32–37. https://doi.org/10.1016/j.procs.2016.05.122 Fourth International Conference on Recent Trends in Computer Science & Engineering (ICRTCSE 2016).

[21] Farida Mohsen, Jiayang Wang, and Kamal Al-Sabahi. 2020. A hierarchical self-attentive neural extractive summarizer via reinforcement learning (HSASRL). *Applied Intelligence* 50 (09 2020). https://doi.org/10.1007/s10489-020-01669-5

[22] Laura Moreno, G. Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2014. Automatic generation of release notes. *Intl. Symp. Found. Softw. Eng.* (01 2014), 484–495.

[23] Laura Moreno, Gabriele Bavota, Massimiliano Di Penta, Rocco Oliveto, Andrian Marcus, and Gerardo Canfora. 2017. ARENA: An Approach for the Automated Generation of Release Notes. 43, 2 (feb 2017), 106–127. https://doi.org/10.1109/TSE.2016.2591536

[24] Ani Nenkova and Kathleen McKeown. 2012. *A Survey of Text Summarization Techniques*. Springer US, Boston, MA, 43–76. https://doi.org/10.1007/978-1-4614-3223-4_3

[25] Dragomir Radev, Sasha Blair-goldensohn, and Zhu Zhang. 2002. Experiments in Single and Multi-Document Summarization. *First Document Understanding Conference* (09 2002).

[26] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2019. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. https://doi.org/10.48550/ARXIV.1910.10683

[27] Ensheng Shi, Yanlin Wang, Lun Du, Junjie Chen, Shi Han, Hongyu Zhang, Dongmei Zhang, and Hongbin Sun. 2022. On the Evaluation of Neural Code Summarization. In *Proceedings of the 44th International Conference on Software Engineering* (Pittsburgh, Pennsylvania) *(ICSE '22)*. Association for Computing Machinery, New York, NY, USA, 1597–1608. https://doi.org/10.1145/3510003.3510060

[28] Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, Joe Davison, Sam Shleifer, Patrick von Platen, Clara Ma, Yacine Jernite, Julien Plu, Canwen Xu, Teven Le Scao, Sylvain Gugger, Mariama Drame, Quentin Lhoest, and Alexander M. Rush. 2020. Transformers: State-of-the-Art Natural Language Processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, EMNLP 2020 - Demos, Online, November 16-20, 2020*, Qun Liu and David Schlangen (Eds.). Association for Computational Linguistics, 38–45. https://doi.org/10.18653/v1/2020.emnlp-demos.6

[29] Weiran Xu, Chenliang Li, Minghao Lee, and Chi Zhang. 2020. Multi-task learning for abstractive text summarization with key information guide network. *EURASIP Journal on Applied Signal Processing* 2020, 1, Article 16 (April 2020), 16 pages. https://doi.org/10.1186/s13634-020-00674-7

[30] Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo, and Lingxiao Jiang. 2022. iTiger: An Automatic Issue Title Generation Tool *(ESEC/FSE 2022)*. Association for Computing Machinery, New York, NY, USA, 1637–1641. https://doi.org/10.1145/3540250.3558934