2022 IEEE International Conference on Software Maintenance and Evolution (ICSME) | 978-1-6654-7956-1/22/$31.00 ©2022 IEEE | DOI: 10.1109/ICSME55016.2022.00015

Automatic Pull Request Title Generation

Ting Zhang, Ivana Clairine Irsan, Ferdian Thung, DongGyun Han, David Lo and Lingxiao Jiang
School of Computing and Information Systems, Singapore Management University
Email: {tingzhang.2019, ivanairsan, ferdianthung, dhan, davidlo, lxjiang}@smu.edu.sg

***Abstract*—Pull Requests (PRs) are a mechanism on modern collaborative coding platforms, such as GitHub. PRs allow developers to tell others that their code changes are available for merging into another branch in a repository. A PR needs to be reviewed and approved by the core team of the repository before the changes are merged into the branch. Usually, reviewers need to identify a PR that is in line with their interests before providing a review. By default, PRs are arranged in a list view that shows the titles of PRs. Therefore, it is desirable to have a precise and concise title, which is beneficial for both reviewers and other developers. However, it is often the case that developers do not provide good titles; we find that many existing PR titles are either inappropriate in length (i.e., too short or too long) or fail to convey useful information, which may result in PR being ignored or rejected. Therefore, there is a need for automatic techniques to help developers draft high-quality titles.**

In this paper, we introduce the task of automatic generation of PR titles. We formulate the task as a one-sentence summarization task. To facilitate the research on this task, we construct a dataset that consists of 43,816 PRs from 495 GitHub repositories. We evaluated the state-of-the-art summarization approaches for the automatic PR title generation task. We leverage ROUGE metrics to automatically evaluate the summarization approaches and conduct a manual evaluation. The experimental results indicate that BART is the best technique for generating satisfactory PR titles with ROUGE-1, ROUGE-2, and ROUGE-L F1-scores of 47.22, 25.27, and 43.12, respectively. The manual evaluation also shows that the titles generated by BART are preferred.

***Index Terms*—Summarization, GitHub, Pull-Request, Mining Software Repositories**

I. Introduction

As an emerging paradigm, pull-based software development has been widely applied in distributed software development [1], [2]. With the pull-based development model, the main repository of the project is not shared for direct modification among *contributors* [3]. Instead, contributors fork or clone the repository and make changes on their own branch. When their changes are ready to be merged into the main branch, they create Pull Requests (PRs); the *core team* (i.e., the *integrators)* then review the changes made in the PRs to make sure that the changes satisfy functional (e.g., no compile error and test failure) and non-functional requirements (e.g., abide to coding convention). They may propose corrections, engage in discussion with the contributors, and eventually accept or reject the changes [4]. PRs are utilized in nearly half of the collaborative projects in GitHub [3]. Pull-request-based development is usually associated with reduced review times and a high number of contributions [5].

Generally, a PR consists of a title, a description (optional), and a set of commits. The PR title and description are designed to help the *readers* (not limited to integrators, but refers to anyone reading the PR) grasp the context and purpose of the PR. Frequently, a PR is linked with one or more issue reports. Issue reports in issue tracking systems (e.g., GitHub issues) are used to keep track of bugs, enhancements, or other requests. Therefore, many PRs contain the identifiers of the linked issues in their titles or descriptions. Since contributors often neglect to write a PR description, the role of the PR titles is significant. For example, 219,909 PRs (16.4%) do not have descriptions among our collected 1,341,790 PRs.

Another common case is that PRs are displayed in a list view by default so that only the title and other meta-information (e.g., author name, tags, and PR id) are available. In the absence of a PR description, a high-quality title becomes more important for readers to understand the intention of a PR. There are other ways to figure out what a PR is about, such as direct interaction with the PR's owner, or checking the details like commit messages and linked issues. However, these are certainly inefficient for software maintenance.

In addition, PRs usually have a fast turnaround: they are either processed fast or left open for a long time without merging to the main branch [1]. In large projects, it is a challenge for integrators to handle a high volume of incoming PRs [3]. Prior research [6] on bug reports has shown that well-written bug reports are more likely to gain the triager's attention and influence on deciding whether a bug gets fixed. Similarly, the quality of PR, such as length of title and description, has a significant impact on PR evaluation latency [7]. Intuitively, PR titles serve as the first criterion for integrators to decide whether certain PRs are relevant to their expertise, which can potentially speed up the review process. Nowadays, the quality of PR descriptions has gained more attention in practice: many Git service providers support software maintainers to use templates to improve the quality of PR descriptions.[1] However, there is no emphasis on helping developers compose high-quality PR titles.

To fill in this gap, we aim for automatic generation of PR titles to compose accurate and succinct PR titles. We formulate the automatic PR title generation task as a one-sentence summarization task. Its goal is to produce a concise and informative target sequence of tokens (*title*), which is a summary of the given *source* sequence of tokens.

As no prior work has been devoted to this task before, we conducted a comprehensive evaluation on the effective-

[1]https://docs.github.com/en/communities/using-templates-to-encourage-useful-issues-and-pull-requests/about-issue-and-pull-request-templates

ness of five state-of-the-art summarization methods on the automatic PR title generation, including both general-purpose and domain-specific summarization methods. Some of these methods are extractive (i.e., they extract the sentences from the source sequence and then concatenate them to form the target sequence), while others are abstractive (i.e., they generate the target sequence with sentences that are different from the original sentences in the source sequence). For *general-purpose* summarization methods, we have identified three approaches: BERTSumExt [8], BART [9], and Text-To-Text Transfer Transformer (T5) [10]. BERTSumExt [8] is an extractive summarization method that utilizes the pre-trained bidirectional encoder representations from Transformers (BERT) [11]. BART [9] and T5 [10] are two large pre-trained Transformer-based architectures, which can be used for abstractive summarization. For *domain-specific* summarization methods, we evaluate two methods, i.e., PRSummarizer [12] originally designed for PR description generation and iTAPE [13] initially designed for issue title generation. Another related paper is by Liu et al. [14] that proposed a Stack Overflow title generation approach; However the underlying model of their method is T5, which is already included in the general-purpose summarization method.

Specifically, in the work, we would like to answer two Research Questions (RQs) to understand the performance of different methods on the PR title generation task:

RQ1: *In terms of automatic evaluation, how do different methods perform on the PR title generation task?*

RQ2: *To what extent can the best performing approaches automatically generate PR titles as developers would do?*

Due to the lack of a suitable dataset, we construct a dataset named PRTIGER (Pull Request Title Generation), which is the first dataset that can be leveraged for PR title generation. Our focus is to help contributors to compose non-trivial PR titles, which aims to help integrators to grasp the main changes made in the PRs. We identified and applied several rules to keep the PR titles that suit the use scenario. In the end, we have 43,816 PR titles belonging to 495 GitHub repositories in PRTiger. The source sequence in the dataset is the concatenation of PR description, commit messages, and linked issue title, with an average length of 114 words. The *target* sequence is the corresponding PR title, with an average length of 7 words.

ROUGE metrics [15], the standard metrics for automatic evaluation of summarization technique effectiveness, are adopted to evaluate model performance. We also conducted a manual evaluation by inviting three evaluators. For each sample, the evaluators were asked to score three titles by reading the source sequence. The titles generated by the automatic methods and the original human-written titles were randomly shuffled. Evaluation criteria include *correctness*, *naturalness*, and *comprehensibility*. The details can be found in Section V-B. The results suggest that BART outperforms the other techniques in terms of both automatic and manual evaluation. To conclude, our main contributions are threefold:

- We introduce the task of automatic PR title generation.
- We construct a dataset named PRTIGER, which consists of 43,816 PRs from GitHub for the PR title generation task. PRTiger is the first benchmark for this task.
- We conduct evaluation of state-of-the-art summarization methods, including extractive (Oracle extraction, BERTSumExt [8]) and abstractive ones (PRSummarizer [12], iTAPE [13], BART [9], and T5 [10]). The evaluation includes both automatic evaluation and manual evaluation from multiple aspects. The results show that BART outperforms other approaches by significant margins.

The remainder of this paper is organized as follows: Section II describes the background, including motivation, the usage scenario, and problem formulation. Section III gives details about the process to construct the PRTiger dataset. We give an introduction to all the summarization methods we evaluated in Section IV. We describe the experimental settings in Section V and present results of our experiments in Section VI. Section VII presents qualitative analysis and threats to validity. We also discuss related work in Section VIII. We conclude this paper and present future directions in Section IX.

II. BACKGROUND

In this section, we first present the motivation and the usage scenario of automatic PR title generation. Next, we discuss about problem formulation and list some challenges.

A. Motivation

A good PR title should accurately summarize what the PR is about. An appropriate PR title would help the PR readers to set their minds on what things they would expect to read from the PR. However, many PR titles failed to accurately and succinctly summarize the PR content.

For example, **Example 1** in Table I has a very short title, with only one word. It does not summarize the PR well. In this example, the PR description is absent. A good PR title would save readers' time from checking the details of commits. The current title fails to make a clear summary of the PR. Generally speaking, PR titles with very few words are unlikely to convey enough information for readers to grasp the PR content.

On the other hand, as admitted by the PR author that they had limited experience in rebase, **Example 2** in Table I shows a relatively long title which failed to concisely summarize the PR as well. In this example, the PR description is taken straight from a static template, which includes several checkboxes. Yet, except for the template itself, the PR author did not write anything else in the description. The current PR title itself is lengthy and does not summarize the changes. Usually, lengthy titles do not succinctly summarize the source sequence and may be written by contributors with limited experience, as shown in Example 2.

These poorly-written PR titles motivate us to resort to an automatic approach to help contributors compose titles. Automatic PR title generation methods would make the process of submitting new PRs and integrating PRs more efficient. They are helpful to contributors as they can reduce contributors' burden on writing. The generated high-quality PR title is then beneficial to integrators as well, as it can potentially save their time and speed up the reviewing process.

TABLE I
PR TITLE EXAMPLES

Example 1	***Example 2***	***Example 3***
Title: *Various.* **Source:** https://github.com/angular/angular/pull/49 **Description:** None **Commit Message:** • *feat(DartWriter): support string interpolation* • *feat(facade/lang): support int* • *refactor(lexer): rename to scanner, use ints, etc.* **Issue:** None	**Title:** *Please let me know if this is a sufficient translation and I can do more... I'm kinda new to rebase, so forgive me if I'm doing this wrong* **Source:** https://github.com/ant-design/ant-design/pull/6370 **Description** *First of all, thanks for your contribution! :-) Please makes sure these boxes are checked before submitting your PR, thank you!* ☐ *Make sure you propose PR to correct branch: bugfix for master, feature for latest active branch feature-x.x.* ☐ *Make sure you follow antd's code convention.* ☐ *Run npm run lint and fix those errors before submitting in order to keep consistent code style.* ☐ *Rebase before creating a PR to keep commit history clear.* ☐ *Add some descriptions and refer relative issues for you PR.* **Commit Message:** • *testing translation* • *adding advanced search* **Issue:** None	**Title:** *Revert "Add godot version in backtrace message"* **Source:** https://github.com/godotengine/godot/pull/30221 **Description:** *Reverts #28572* *The idea is good, but we can't use the configurable crash handler message to display the version string, it should be hardcoded in the crash handler print code for each platform instead.* **Commit Message:** *Revert "Add godot version in backtrace message"* **Issue:** None

B. Usage Scenario

The considered usage scenario in our work could be stated as follows: When a contributor is drafting a new PR, an automatic approach can help them compose a title that contains an overview of the changes made in the PR. Ideally, the PR title should cover the information from commit messages and also the issue information if the PR is resolving or is linked to any issue. This automatic title generation approach will save contributors' time from thinking about how to write a succinct summary for their PR contents, so they can focus on writing a detailed description.

C. Problem Formulation

We formulate the PR title generation task as a one-sentence summarization task [13]. For simplicity, we refer to a concatenation of PR descriptions, commit messages, and linked/resolved issue titles as a *source* sequence, and the PR title as a *target* sequence in the rest of our paper. We will also use *target* and *title* to refer the PR title interchangeably. For a PR title generation model, the input is the source sequence, and it is supposed to generate a PR title, which is the one-sentence summary of the source sequence.

In this work, we focus on generating non-bot written and non-*trivial* PR titles. *Trivial* PR titles are those that are authored by humans and can be written with minimal effort since PRs handle common tasks. One such example is **Example 3** in Table I. In this example, the PR description contains the ID and the description of PR that needs to be reverted. The title *Revert "Add godot version in backtrace message"* is sufficient to understand the purpose of this PR. The type of PR titles does not ask for summarization ability. A concatenation of a keyword (i.e., `revert`) and the title of the PR that needs to be reverted would suffice and considered to be a common and good practice. For bot-written titles, contributors do not need to bother writing titles. Thus, for bot-written and trivial titles, there is no need for an automatic approach to help compose PR titles.

The PR title generation task is similar to PR description generation [12] and title generation for issues [13] and Stack Overflow posts [14]. However, the PR title generation task has its own unique challenges: (1) Compared to the title generation task for issues/Stack Overflow posts, in the PR title generation task, there is a semantic gap between different resources, i.e., PR description, commit messages, and issues. In particular, the two resources of our input sequence: the linked issue and the PR description are generally written by different authors (issue reporter and issue triager, respectively); (2) Compared to the PR description generation task, the target sentence (PR title) is quite short and it has to be as concise and precise as possible. The four tasks are complementary to each other and help reduce the workload of developers in different ways.

III. Benchmarking Dataset Building

As there is no existing dataset for PR title generation, we decide to build the first benchmark dataset for this task. In this work, we experimented with GitHub data due to its popularity. Although Liu et al. [12] shared a dataset for the PR description generation task, they did not include the titles of PRs in their dataset. In addition, their dataset only contains engineered Java projects, which may lack diversity. Therefore, a new dataset containing PR titles gathered from various programming language repositories is needed. We first collected the data; next, we filtered out PRs which did not belong to the usage scenario we focused on in this work. Then, we filtered out the content from PRs which do not help to generate accurate and succinct PR titles.

A. Data Collection

To collect PR data from GitHub, we firstly got 7 repository lists: Top-100 most-starred and Top-100 most-forked repositories (regardless of programming language); Top-100 most-starred repositories that are written primarily in one of the following languages: JavaScript, Python, Java, C,

TABLE II
OUR TRIVIAL PR TITLE PATTERNS, AND EXAMPLE PR TITLES FROM THE COLLECTED DATASET.

Starts with (lowercased)	Example Title
automated cherry pick of	*Automated cherry pick of #12022 #13148 upstream release 1.0*
merge to	*Merge to live part 2 on 4-27*
revert	*Revert "Add log_only to debug messages"*
rolling up	*Rolling up changes to staging from master*
rolling down	*Rolling down changes from staging to master*
roll engine	*Roll engine dart roll 20180920*
rollup of	*Rollup of 5 pull requests*
roll plugins	*Roll Plugins from 6d8ea78c5da1 to 361567b9189c (4 revisions)*
update live with current master	*Update live with current master*

TABLE III
DATA STATISTICS ON OUR COLLECTED PULL REQUESTS

With < 2 or > 20 commits	With non-ASCII characters	Authored by bot	With trivial titles
1,147,734	16,396	1,642	111,780

PRs Left	Total PRs Collected
51,753	1,341,790

and C++.[2] The number of stars and the number of forks are two different metrics in GitHub: The number of star means how many people click on the repository to show their appreciation. A fork is a copy of a repository and the number of forks indicates how many people copied this project. The Top-100 most-starred and most-forked repositories cover a wide range of programming languages, e.g., Shell (`ohmyzsh/ohmyzsh`), Go (`kubernetes/kubernetes`), and TypeScript (`microsoft/vscode`). Given the 7 lists have common repositories, after removing redundancies, in total we crawled 578 distinct repositories. We collected the PRs from each repository using GitHub GraphQL API.[3] For each repository, we only kept the merged PRs that were published up to December 31, 2021. Given a merged PR published before the year 2022, we retrieved its title, description, commit messages, and linked issues. In total, we collected 1,341,790 merged PRs from the 578 GitHub repositories.

B. Data Preprocessing

Selecting PRs. For each PR, to better simulate the scenario when a contributor opens a new PR, we first removed the commits submitted after the PR was created. Then, following Liu et al. [12], we filtered out the PRs which have less than two commits or more than 20 commits. As Liu et al. pointed out, we can directly use the commit message as the PR title if a PR only contains one commit, and a PR with too many commits is usually used for synchronization purpose instead of being a typical contribution from developers. We also removed the PRs (1) containing non-ASCII characters; (2) authored by bots. In addition, we also filtered out the PRs which contain *trivial* titles, where automatic methods for generating PR titles are not needed. We mainly identified the following four types of trivial titles: (1) *Recurrent titles.* Table II shows the templates which occurred prominently in our collected PRs. If any PR title starts with these patterns, we exclude it. (2) *Too short or too long titles.* Following the dataset building rules used by iTAPE [13], we excluded the titles with less than 5 words or more than 15 words. Similar to Chen et al. [13], we observe that PR titles having 5-15 words are of reasonably appropriate length to precisely and succinctly describe key ideas. (3) *Titles with limited overlap with the source sequence.* If 20% of the words in the title were not present in the source sequence, we considered the title not to be a good summary of the source sequence and therefore excluded it from the dataset. (4) *Titles were copied from the source sequence.* We first lowercased both the title and the source sequence. If the title can be exactly matched to the description or concatenation of the commit messages, we removed the PR from the dataset as we consider this PR as a bad example to train the model.

After selecting PRs based on the above criteria, we have 51,753 PRs left in the dataset. Table III shows the statistics of our collected PRs.

Cleaning the selected PRs. We followed iTAPE [13] to remove tags in the PR titles. We also followed PRSummarizer [12] to (1) remove checklists in the source sequence; and (2) remove identifiers in both source and target sequence. We also added extra steps: (1) Removing PR templates in the source sequence. We first queried through the GitHub API to find out whether a repository provided a PR template for contributors to compose a PR description. If there was a PR template, we saved the template string. In our dataset, 214 out of 495 repositories provided a PR template at the time we called the GitHub API. To remove the template information from each PR, we first split the PR description into lines. Then, for each line in the source sequence, if it can be matched exactly to the template string, we removed it. Otherwise, we kept the lines. By doing this, we reduced the noise in the source sequence. (2) Removing automatically generated commit messages in the source sequence. We observed that many commit messages only convey the *merge branch* information, e.g., *Merge branch 'master' into tst-indexing-16*. We used the following four regular expressions to remove the automatically generated commit messages: (1) `merge .*? branch .*? into` (2) `merge branch .*? into` (3) `merge pull request \#` and (4) `merge branch \'`.

After applying the pre-processing steps to our data, we further excluded PRs which have less than 30 words or more than 1,000 words. In the end, we have a dataset *PRTiger* consisting of 43,816 PRs for experiments. We split PRTiger into training, validation, and test sets with a ratio of 8:1:1. Table IV shows the number of instances and the average word count in the train, validation, and test set respectively.

[2]The detailed repository lists are available here: https://github.com/EvanLi/Github-Ranking/blob/f4cf5694eaed7ad1ee59425d7c0dcf9f3e8511f9/Top100

[3]https://docs.github.com/en/graphql

TABLE IV
DATA STATISTICS ON DIFFERENT SPLITS

	Train		Validation		Test	
	Source	**Target**	**Source**	**Target**	**Source**	**Target**
Instance #	35,052	35,052	4,382	4,382	4,382	4,382
Avg Word #	114	7	114	7	112	7

IV. SUMMARIZATION METHODS

In this section, we elaborate on the summarization methods evaluated in this work. Summarization methods can be broadly categorized into two groups, i.e., *extractive methods* and *abstractive methods* [16]. An *extractive* summarization method extracts sentences from the source sequence and then concatenates them to form the target sequence. In contrast, an *abstractive* summarization method represents the source sequence in an intermediate representation. It then generates the target sequence with sentences that are different from the original sentences in the source sequence [17]. We experimented with state-of-the-art extractive and abstractive methods. Their details can be found as follows:

A. Extractive Methods

We experimented with two extractive summarization methods, i.e., oracle extraction and BertSumExt [8].

Oracle Extraction: This is not a *real* approach. Instead, it can be viewed as an upper bound of extractive summarization methods or serve as a measure to gauge the capability of other extractive summarization methods. Oracle extraction scores have been commonly used in summarization literature for comparison purpose [18], [19]. It may have different variants depends on the specific task setting. In our work, oracle extraction selects the sentence from the source sequence that generates the highest ROUGE-2 F1-score compared to the original title. We firstly split the PR description, commit messages, and issue titles into sentences. Next, we computed the ROUGE-2 F1-score of each sentence with the original PR title. We selected the sentence with the highest ROUGE-2 F1-score as the generated title by this method. Since oracle extraction needs the original title as a reference, it cannot be applied in practice. However, it can be used for comparison to understand other extractive methods' performance.

BertSumExt [8]: BertSumExt is built on top of BERT-based encoder by stacking several inter-sentence Transformer layers. Specifically, for each sentence in the source sequence, BertSumExt represents each sentence with the vector of the *i*-th `[CLS]` symbol from the top layer. Then, several inter-sentence Transformer layers are then stacked on top of BERT outputs, to capture document-level features for extracting sentences as the summary. BertSumExt achieves better results with less requirement on the model, compared to other models that enhance the summarization via the copy mechanism [20], reinforcement learning [21], and multiple communicating encoders [22]. Identical to what we did in oracle extraction, we also split PR description, commit messages, and issue titles into sentences. BertSumExt will then give a score to each sentence based on the suitability of becoming a summary. BertSumExt was initially evaluated on three single-document news summarization datasets, which could be summarized in a few sentences. In the original implementation, BertSumExt chooses the sentences with Top-3 highest scores as a summary. As the PR title generation task was formulated as a one-sentence summarization task, we take the Top-1 sentence as the generated PR title.

B. Abstractive Methods

The two most-similar works are identified, i.e., PRSummarizer [12] and iTAPE [13]. As they did not directly choose the sentence from the source sequence, we also categorized them into abstractive methods. Besides, we applied BART [9] and T5 [10], which are state-of-the-art method for text summarization. (we put the exact model version of the Hugging Face `transformers` library that we use [4] in parentheses):

PRSummarizer [12]: This text summarization model was designed to automatically generate PR descriptions from the commits submitted with the corresponding PRs, which is a sequence-to-sequence (Seq2seq) learning task. The underlying model of PRSummarizer is the attentional encoder-decoder model [23]. Besides, PRSummarizer can handle two unique challenges, i.e., out-of-vocabulary (OOV) words and the gap between the training loss function of Seq2seq models and the discrete evaluation metric ROUGE. Especially, PRSummarizer copes with OOV words by integrating the pointer generator [24]. The pointer generator either selects a token from the fixed vocabulary or it will copy one token from the source sequence at each decoding step. To minimize the gap between the loss function and the ROUGE metrics, PRSummarizer also leverages a reinforcement learning (RL) technique named self-critical sequence training [25] and adopts a particular loss function named RL loss [21].

iTAPE [13]: iTAPE uses a Seq2seq based model to generate the issue title using the issue body. Specifically, iTAPE adopts the attentional RNN encoder-decoder model. In addition, to help the model process the low-frequency human-named tokens effectively, iTAPE firstly inserts additional "tag tokens" before and after each human-named token, i.e., identifiers and version numbers. These tag tokens are added to the issue body to indicate their latent semantic meanings. Furthermore, iTAPE adopts a copy mechanism [20], which allows the model to copy tokens from the input sequence. The underlying architecture is the pointer-generator [24], a commonly used abstractive summarization approach before the dominant usage of pre-trained models.

BART [9] (`facebook/bart-base`): is a Seq2seq autoencoder based on a standard Transformer [26] architecture. The pre-training process of BART consists of two stages: (1) corrupt the input text by using an arbitrary noising function and (2) a Seq2seq model is learned to reconstruct the

[4]https://huggingface.co/models

original text by minimizing the cross-entropy loss between the decoder output and the original sequence. A number of noising approaches are evaluated in the BART paper. The best performance is achieved by adopting both the noising methods, i.e., (1) randomly shuffling the order of the original sentences, and (2) applying an in-filling scheme, where a single mask token is used to replace arbitrary length spans of text (including zero length). BART was pre-trained with the same data as RoBERTa [27], i.e., 160GB of news, books, stories, and web text. BART achieves new state-of-the-art results on several text generation tasks, including abstractive dialogue, question answering, and summarization tasks.

T5 [10] (`t5-small`): T5 is a pre-trained language model which aims to convert all NLP tasks into a unified text-to-text-format where the input and output are always text strings. The advantage of this T5 text-to-text framework is that we can use the same model, loss function, and hyper-parameters on any NLP task. T5 is also based on the Transformer architecture. Similar to BART, T5 was pre-trained on a masked language modeling objective: contiguous spans of token in the input sequence are replaced with a mask token and the model is trained to reconstruct the masked-out tokens. Unlike BART, T5 was pre-trained with the Colossal Clean Crawled Corpus (C4) dataset, which consists of 750GB of English text from the public Common Crawl web scrape. T5 was reported to achieve state-of-the-art results on many benchmarks including summarization, question answering, and text classification.

V. Experimental Settings

This section describes the relevant design and settings of our study. We list two research questions and the evaluation metrics, and briefly describe the implementation details.

A. Research Questions

We would like to empirically evaluate the performance of different approaches on the automatic PR title generation task. The study aims to answer the following RQs:

RQ1: *In terms of automatic evaluation, how do different methods perform on the PR title generation task?* Although there is no prior work on automatically generating PR titles, we choose the approaches from the two closest works [12], [13] as the baselines. We used the implementations of these two approaches on our task. For comparison, we also evaluated the state-of-the-art general-purpose extractive and abstractive summarization techniques.

RQ2: *To what extent can the best performing approaches automatically generate PR titles as developers would do?* Other than providing the results on automatic evaluation, we also conducted a manual evaluation. Given the expense to run manual evaluation, we only evaluated the two best-performing methods on the automatic evaluation. We invited three annotators who are not an author of this paper.

B. Evaluation Metrics

Automatic Evaluation Following the prior works [12], [13], we use Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [15] to measure the quality of generated summaries for the summarization task. ROUGE-N measures the overlap of n-grams [15] between the model-generated summary and the reference summary. The formulas to calculate ROUGE-N can be shown as follows:

$$R_{ROUGE-N} = \frac{Count(overlapped\_N\_grams)}{Count(N\_grams\_in\_reference\_summary)}$$

$$P_{ROUGE-N} = \frac{Count(overlapped\_N\_grams)}{Count(N\_grams\_in\_generated\_summary)}$$

$$F1_{ROUGE-N} = 2 \times \frac{R_{ROUGE-N} \times P_{ROUGE-N}}{R_{ROUGE-N} + P_{ROUGE-N}}$$

As the variable names suggest, R (recall) measures the percentage of the N-grams in the reference summary that has been covered by the generated summary, and P (precision) presents the percentage of N-grams in the generated summary that is, in fact, relevant or needed. F1-score of the ROUGE scores is used to represent and give an equal importance between recall and precision. In this task, we report the precision, recall, and F1-score of ROUGE-N (N=1,2) and ROUGE-L from each method. ROUGE-1, ROUGE-2, and ROUGE-L are commonly used in the literature to understand the summary quality [12], [13]. ROUGE-1 and ROUGE-2 measure the overlap of uni-grams (1-grams) and bi-grams (2-grams), respectively. A uni-gram consists of a single word, while a bi-gram consists of two consecutive words. Instead of n-grams, ROUGE-L measures the longest common subsequence between the reference summary and the generated summary. We treat F1-score of ROUGE as the main summarization performance measure. We first get the generated summaries from each approach. For the ROUGE scores calculation, we adopt the metric implemented in Hugging Face `datasets` library [28].

Manual Evaluation In addition to automatic evaluation, we also conducted a manual evaluation to better understand and evaluate the quality of titles generated by different approaches. Since ROUGE scores are calculated based on the overlap of n-grams, the generated summaries may be semantically incorrect, even with very high ROUGE scores. This is due to the ROUGE scores limitation which only measures the lexical similarity between two sentences. Hence, it cannot

TABLE V
An example PR with the original title and the titles generated by BART and T5.
https://github.com/grpc/grpc/pull/1655

Source	<desc> this fixes #1551. servers now respond to unparseable arguments with the invalid_argument status and the text of the deserialization error, instead of crashing. </desc> <cmt> added failing tests for server bad argument handling </cmt> <cmt> fixed server to handle invalid arguments without breaking </cmt> <iss> node rpc server cannot recover from malformed requests </iss>
Original Title	handle invalid arguments sent to the node server
BART	fix server bad argument handling
T5	fix server to handle invalid arguments without breaking node

gauge the comprehensibility of the summaries generated by the model. [29] Thus, we sampled 150 PRs from the test set. With this sample size, we can maintain the confidence level of 92% with an 8% margin of error. Three evaluators were invited to give the quality scores to the PR titles generated by the two best approaches and the original titles written by developers. All of the evaluators have more than 6-year experience in programming and more than 5-year experience using GitHub. They have a basic understanding of how pull-based software development works (i.e., they are experienced in making changes, pushing commits, writing issue reports, and opening PRs).

The techniques used to generate titles are hidden from the evaluators; they cannot judge based on the bias of knowing the authorship. For each sample, evaluators were provided with the *source* sequence along with three *titles*: two of the titles are generated by the two best-performing approaches, i.e., BART and T5, and the rest is the original title. We randomly shuffled the order of the three titles. To help the evaluators read the source sequence clearly, we use `<desc></desc>`, `<cmt></cmt>`, and `<iss></iss>` to enclose description, commit messages, and linked issue titles, respectively. One sample source sequence can be seen in Table V. Evaluators were required to read through the *source* sequence and the three titles. They were asked to score the three titles (1 - very poor; 5 - very good) with regards to the following aspects:

- **Correctness:** To which extent, do you think the title *correctly* summarize the source sequence?
- **Naturalness:** To which extent, do you think the title is resembling a human-written title?
- **Comprehensibility:** To which extent, do you think the title is easy to *understand*?

Additionally, they were also required to rank the three titles. Their personal preference for these titles does not necessarily base on the three criteria listed above. If the titles are the same, they can rank two titles with the same rank. Otherwise, they must give different ranks for each title.

C. Implementation Details

We run all the experiments with NVIDIA Tesla V100 GPUs. For iTAPE, we preprocessed the identifier and version numbers with the scripts provided by the authors.[5] For PRSummarizer, we changed the vocabulary size from 50k to 200k to handle the OOV issue. Given BART (`BART-base`, which contains 140 millon parameters[6]) has more parameters than T5 (`T5-small`, which contains 60 million parameters[7]), we run BART with the batch size of 4; while 8 for T5. All the remaining hyper-parameters were left as the default values. The detailed default values can be found in our replication package.[8]

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

[6]https://github.com/pytorch/fairseq/blob/main/examples/bart/README.md#pre-trained-models

[7]https://github.com/google-research/text-to-text-transfer-transformer#released-model-checkpoints

[8]https://github.com/happygirlzt/ICSME-PRTiger

VI. Experimental Results

A. RQ1: Comparison on Automatic Evaluation

To investigate how different approaches perform on the PRTiger dataset, we firstly analyze model performance with ROUGE metrics. Table VI shows the ROUGE scores from the six approaches.

Firstly, looking at the length of generated titles, we can observe that the extractive approaches produce longer titles than the abstractive approaches. The average length of the titles generated by extractive approaches highly relies on the length of sentences selected from the dataset. Although we set the extractive approaches to select a single sentence as a title, they select longer titles. All the abstractive methods generate titles in the average length of 6-8 words. The length range is similar to the average length across all the data splits (See Table IV). It indicates that abstractive methods generate titles with an appropriate length, as it is capable of generating titles with the average length similar to the average length of the original titles.

Serving as the upper bound of extractive summarization approaches, Oracle Extraction gives the highest ROUGE-1, ROUGE-2, and ROUGE-L F1-scores since it relies on the original title. In comparison, BertSumExt gave ROUGE-1, ROUGE-2, and ROUGE-L F1-score of 37.64, 18.48, and 33.94. These scores were 20%, 38.9%, and 22.7% lower than the Oracle Extraction. It suggests that, in practice, PR titles have a considerable amount of overlap with the source sequence. Yet, BertSumExt is not capable of selecting the correct sentence every time.

Among the four abstractive approaches, BART and T5 achieved better performance than the two other abstractive approaches. It demonstrates the power of pre-training in the PR title generation. BART and T5 were both pre-trained with large general-purpose corpora. Unlike these two approaches, PRSummarizer and iTAPE were trained solely on the PRTiger data, where PRSummarizer produces a better performance than iTAPE. Considering PRSummarizer was originally proposed for PR description generation, the data they used to evaluate their approach has similar characteristics as PRTiger. In addition, both PRSummarizer and iTAPE adopt the pointer generator [24]. It suggests the importance of RL-loss in PRSummarizer, which is used to minimize the gap between the loss function and the ROUGE metrics.

Comparing extractive and abstractive methods, BART shows an on-par performance as the Oracle Extraction, which indicates that BART is capable of capturing key points from the source sequence and generating precise PR titles. BART outperforms the second-best approach T5 by 12.2%, 18.1%, and 12.1%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Although T5 gives worse performance than BART, it still outperforms the other approaches, i.e., PRSummarizer, by 10.9%, 18.9%, and 10% with regards to ROUGE-1, ROUGE-2, and ROUGE-L F1-scores, respectively. Interestingly, BertSumExt gives an on-par performance as PRSummarizer and it outperforms iTAPE. Given the words

TABLE VI
AUTOMATIC EVALUATION RESULTS ON THE TEST DATASET

Approach	Avg. Length	ROUGE-1			ROUGE-2			ROUGE-L		
		Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Extractive										
– **Oracle Extraction**	13	47.61	55.74	46.97	32.51	34.04	30.25	44.88	51.33	43.91
– **BertSumExt**	13	36.18	44.95	37.64	18.18	21.71	18.48	32.76	40.28	33.94
Abstractive										
– **PRSummarizer**	6	41.4	36.68	37.91	20.06	17.26	17.99	38.22	33.83	34.98
– **iTAPE**	8	31.63	34.62	32.23	12.67	13.98	12.91	28.71	31.51	29.31
– **BART**	7	50.03	46.98	47.22	27.15	24.96	25.27	45.71	42.85	43.12
– **T5**	7	44.88	42.15	42.06	23.22	21.3	21.39	41.09	38.49	38.46

TABLE VII
PERFORMANCE COMPARISON WITH OTHER SUMMARIZATION TASKS: ISSUE TITLE GENERATION (CHEN ET AL. [13]), PR DESCRIPTION GENERATION (LIU ET AL. [12]), AND LAY LANGUAGE SUMMARIZATION OF BIOMEDICAL SCIENTIFIC REVIEWS (GUO ET AL. [30])

Method	ROUGE-1 F1	ROUGE-2 F1	ROUGE-L F1
BART in our work	47.22	25.27	43.12
Chen et al. [13]	31.36	13.12	27.79
Liu et al. [12]	34.15	22.38	32.41
Guo et al. [30]	53.02	22.06	50.24

in the PR, titles are not necessarily contained in the source sequence, the PR title generation task is naturally an abstractive summarization task. However, as an extractive approach, BertSumExt shows relatively good performance. It indicates that, in practice, developers may draft titles based on existing sentences, either in PR description or commit messages.

Other than comparing different approaches on the automatic PR title generation task solely, we show the ROUGE F1-scores from other related tasks. Chen et al. [13] work on the issue title generation task, where the proposed approach is named iTAPE and is evaluated in our work. Liu et al. [12] work on the PR description generation task, where the proposed approach is named PRSummarizer and is evaluated in our work as well. Guo et al. [30] work on automated generation of lay language summaries of biomedical scientific reviews. The ROUGE F1-scores from the best performing method of these three tasks are present in Table VII. According to the result, we can find that BART in our work achieves a comparable-level of performance on the automatic PR title generation task.

Automatic Evaluation The fine-tuned BART and T5 outperform all the other approaches. BertSumExt is on par with the two existing approaches (PRSummarizier and iTAPE). The best-performing approach, BART, outperforms the second-best approach by 12.2%, 18.1%, and 12.1%, in terms of ROUGE-1, ROUGE-2, and ROUGE-L F1-scores.

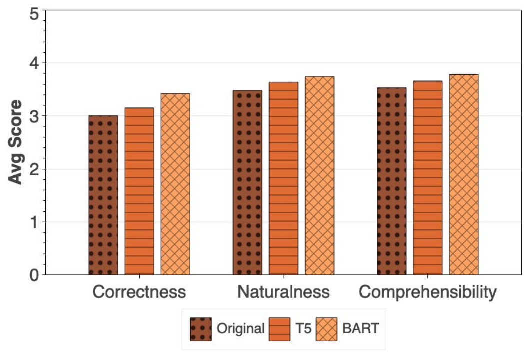

Fig. 1. Average scores from three evaluators

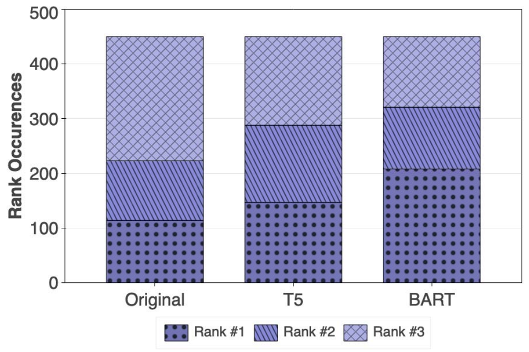

Fig. 2. Rank occurrences for the original titles and the titles generated by BART and T5.

B. RQ2: Comparison on Manual Evaluation

Figure 1 shows the average scores regarding three aspects from the three evaluators. BART is better than T5 in the three aspects, which is in line with the automatic evaluation results with ROUGE metrics. Surprisingly, we can see that BART and T5 show higher scores than the original titles in all three perspectives. It indicates that the titles generated by automatic methods are more acknowledged than the titles written by PR authors. Across the three aspects, we can find that all the three approaches receive the lowest average scores of *correctness*. They achieve slightly higher average scores

TABLE VIII
AN EXAMPLE PR WITH THE ORIGINAL TITLE AND THE GENERATED TITLES FROM FIVE APPROACHES:
HTTPS://GITHUB.COM/ILUWATAR/JAVA-DESIGN-PATTERNS/PULL/1066

Source Sequence	reduces **checkstyle errors for** patterns: **api-gateway lazy-loading leader-election** changes involved java docs reordering imports indentations line length issues reduces **checkstyle errors** in **lazy-loading** reduces **checkstyle errors** in **leader-election** reduces **checkstyle errors** in **api-gateway**
Original Title	**resolves checkstyle errors for api-gateway, lazy-loading, leader-election**
BART	**resolves checkstyle errors for api-gateway lazy-loading leader-election**
T5	reduces **checkstyle errors for** patterns
BertSumExt	reduces **checkstyle errors for** patterns : **api - gateway lazy - loading leader - election** changes involved java docs reordering imports indentations line length issues
PRSummarizer	reduces **checkstyle errors for** patterns: **api-gateway**
iTAPE	resolves **checkstyle errors for** patterns : **api-gateway lazy-loading**

regards *comprehensibility* than *naturalness*. These results show that although PR titles read like written by humans and are easy to understand, they may not be correct enough from the evaluators' perspectives.

Figure 2 shows that for each rank, the percentage of different approaches. Although the ranking is a relative order, it is allowed to be subjective and solely based on the evaluators' personal preference. For the best ones, we can find that BART has been listed as the best one the most times. T5 comes the second. Again, surprisingly, the original title is less preferred.

While a larger-scale study is required, our work provides preliminary evidence that automatically generated PR titles are readable and comprehensible to PR readers, i.e., have higher scores in terms of correctness, naturalness, and comprehensibility. In the sampled PR titles, automatically generated PR titles are preferred over the original titles in most cases.

Manual Evaluation The PR titles generated by BART gained the highest scores on correctness, naturalness, and comprehensibility. T5 performs the second. They are both preferred more than the original titles.

VII. DISCUSSION

A. Qualitative Analysis

Composing PR titles is a non-trivial task, as it requires both summarization ability and domain knowledge. Moreover, PR titles are generally short. Here, we would like to seek a qualitative understanding of the performance difference between different approaches.

Automatic Evaluation. In Table VIII, we showcase one example of titles produced by each approach and the original title along with the source sequence. Note the source sequence is the cleaned version used for the models to generate titles. We use boldface to highlight the common part between the text and the original title. All the approaches can correctly generate `checkstyle errors for`, which occurs several times in the source sequence. However, all the approaches except BART generate the first word as `reduces`, and they also generate `pattern`. It is natural, as the first sentence in the source sequence contains `reduces` and `pattern` as well. It is interesting to see that BART does not directly take the first sentence. Besides, the body does not even have the word `resolve`. We checked the dataset and found that several PR titles in the same repository (`iluwater/java-design-pattern`) followed the same name style: `resolves checkstyle errors for`. The following text differs among PRs. From this example, we can find that BART can learn this title style instead of simply choosing the first sentence. BART does not only use exactly the same words from the source sequence. Instead, it could correctly generate the words which are not present in the source sequence. It shows BART has the promising ability to be adopted in the automatic PR title generation task.

Manual Evaluation. In Table V, we show an example from our sampled 150 PRs. All the evaluators indicate their preference among the three titles as: BART > T5 > Original. Firstly, this PR contains description, two commit messages and linked issue. The original title uses `handle`, which only appears in one of the commit messages. Besides, the phrase `sent to` does not exist in the source sequence. In comparison, the generated title from BART and T5 used `fix`, which occur twice in the source sequence and all the words in the generated titles are present in the source sequence. The title generated by BART is shorter and summarizes the source sequence well, while the title generated by T5 is longer and covers more words from the source sequence.

We investigated PR titles with high comprehensibility scores (i.e., all evaluators gave a score of >=4 for comprehensibility). We found that a PR title is considered to be more comprehensible when: (1) It covers multiple sources of information, e.g., sentences in the PR description and commit messages. (2) It explicitly states that it fixes an issue or adds a feature (e.g., usually starts with the word "fixed" or "added"). We also investigated PR titles with low comprehensibility scores (i.e., all evaluators gave a score <=3 for comprehensibility). We found that a PR title is considered to be less comprehensible when: (1) the PR contains many commits but the PR title only takes information from a few commit messages. (2) The PR title has little connection with the PR description.

B. Threats to Validity

Threats to internal validity relate to the experimental bias. Following the prior works in summarization studies [12], [13], we adopted both automatic evaluation (i.e., ROUGE metrics) and manual evaluation. Like other manual-involved evaluations, our experimental results may be biased or inconsistent. To mitigate this issue, we recruited 3 evaluators with 6+ years of computer programming and 5+ years of experience in using GitHub. They are familiar with the issue and code review mechanism in GitHub.

Moreover, during the dataset building process, although we have performed the selection heuristics, the remaining PR titles that we use as ground truth for automatic evaluation may not be the ideal ones. We use them as proxies of the ideal reference titles for scalability reasons. There are a total of 43,816 titles in our experimental dataset and crafting the best possible reference titles for all of them manually is not practically feasible. We address this limitation of the automatic evaluation by manual evaluation; each of these evaluations has its own limitations: automatic (quality of reference titles) vs. manual (limited number of titles are considered).

Threats to external validity relate to whether our findings can be generalized to other datasets. To alleviate the external threats, we consider repositories from different programming languages diverse. We also considered two metrics in GitHub: Top-100 most-starred and Top-100 most-forked repositories. Although we only included the repositories from GitHub, we do not identify a large difference between repositories from GitHub and those from other git service providers [31]. Therefore, we consider the external threat to be minimal.

VIII. Related Work

In this section, we review the two lines of research most related to our work: understanding pull requests and automatic software artifact generation.

Understanding Pull Requests. Pull-based software development has attracted more and more interest in research. Some works focus on empirically understanding PRs. Gousios et al. conducted two surveys to analyze the work practices and challenges in pull-based development from the integrator's [3] and contributor's [4] perspectives, respectively. In the first piece of work, they found that integrators successfully use PRs to solicit external contributions. Integrators concern with two major factors in their daily work, i.e., (1) quality: both at the source code level and tests (2) prioritization: typically they need to manage a large number of contribution requests at the same time. In the later one, they performed an exploratory investigation of contributors to projects hosted on GitHub. Their findings include, but are not limited to, the fact that contributors are very interested in knowing project status for inspiration and to avoid duplicating work, but they are not actively trying to publicize the PRs they are preparing.

Other works are interested in solving PR-related tasks. Yu et at. [32] studied the reviewer recommendation for PRs to improve the PR evaluation process. They found that traditional approaches (e.g., SVM-based) for bug triage are also feasible for PR reviewer recommendations. They also found that combining the social factors (e.g., common interests among developers) and technical factors (e.g., developer's expertise) is efficient to recommend reviewers. Jiang et al. [33] studied the tag recommendation for PRs. They proposed FNNRec, which uses a feed-forward neural network to analyze titles, descriptions, file paths, and contributors, to help developers choose tags. These prior works motivate our work to automatically generate high-quality PR titles to benefit both developers and reviewers. Besides, good PR titles can also be helpful for downstream tasks which utilize PR titles.

Automatic Software Artifact Generation. Automatic software artifact generation has gained emerging interests in SE research. Existing works range from automatically generating release notes [34], bug reports [35], to commit messages [36]. Moreno et al. [34] introduced an automatic approach to generate release notes. The proposed approach extracts changes from the source code, summarizes them, and integrates the summary of changes with information from versioning systems and issue trackers. Li et al. [35] proposed an unsupervised approach for bug report summarization. The approach leverages an auto-encoder network with evaluation enhancement and predefined fields enhancement modules. Xu et al. [36] proposed CODISUM to address the two limitations of prior commit message generation task, i.e., ignoring the code structure information and suffering from the OOV issue. Specifically, to better learn the code changes representations, they first extract both code structure and code semantics from the source code changes, and then jointly model these two information sources. Moreover, they adopted a copy mechanism to mitigate the OOV issue. We also mention iTAPE, an issue title generation approach in Section IV. Our work and these works are complementary. We all aim to promote automatic generation in SE, while concentrating on different aspects.

IX. Conclusion and Future Work

In this paper, we propose the task of automatic generation of PR titles. To facilitate the research on this task, we construct a dataset named PRTiger, which consists of 43,816 PRs from 495 GitHub repositories. We conducted both automatic and manual evaluations on the state-of-the-art summarization approaches for the automatic PR title generation task. The experimental results indicate that BART is the most capable technique for generating satisfactory PR titles with ROUGE-1, ROUGE-2, and ROUGE-L F1-scores of 47.22, 25.27, and 43.12, respectively. Manual evaluation also shows that the titles generated by BART are preferred.

We believe that our work opens up many interesting research opportunities. To name a few, one possible research direction is to consider the characteristics of PR data to propose a domain-specific pre-trained model. Domain-specific models are promising, such as BERTweet [37], which is pre-trained on English tweets, outperforms the strong baseline RoBERTa$_{\text{base}}$ [27] on three Tweet NLP tasks. Additionally, to further improve the PR title generation performance, another direction is to leverage the hierarchical code structure information from code changes.

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] G. Gousios, M. Pinzger, and A. v. Deursen, "An exploratory study of the pull-based software development model," in *Proceedings of the 36th international conference on software engineering*, 2014, pp. 345–355.

[2] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *2015 IEEE/ACM 12th Working Conference on Mining Software Repositories*, 2015, pp. 367–371.

[3] G. Gousios, A. Zaidman, M.-A. Storey, and A. Van Deursen, "Work practices and challenges in pull-based development: The integrator's perspective," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1. IEEE, 2015, pp. 358–368.

[4] G. Gousios, M.-A. Storey, and A. Bacchelli, "Work practices and challenges in pull-based development: the contributor's perspective," in *2016 IEEE/ACM 38th International Conference on Software Engineering (ICSE)*. IEEE, 2016, pp. 285–296.

[5] J. Zhu, M. Zhou, and A. Mockus, "Effectiveness of code contribution: From patch-based to pull-request-based tools," in *Proceedings of the 2016 24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, 2016, pp. 871–882.

[6] P. J. Guo, T. Zimmermann, N. Nagappan, and B. Murphy, "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*, 2010, pp. 495–504.

[7] Y. Yu, H. Wang, V. Filkov, P. Devanbu, and B. Vasilescu, "Wait for it: Determinants of pull request evaluation latency on github," in *2015 IEEE/ACM 12th working conference on mining software repositories*. IEEE, 2015, pp. 367–371.

[8] Y. Liu and M. Lapata, "Text summarization with pretrained encoders," *arXiv preprint arXiv:1908.08345*, 2019.

[9] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019.

[10] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu, "Exploring the limits of transfer learning with a unified text-to-text transformer," *arXiv preprint arXiv:1910.10683*, 2019.

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

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

[13] S. Chen, X. Xie, B. Yin, Y. Ji, L. Chen, and B. Xu, "Stay professional and efficient: automatically generate titles for your bug reports," in *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*. IEEE, 2020, pp. 385–397.

[14] K. Liu, G. Yang, X. Chen, and C. Yu, "Sotitle: A transformer-based post title generation approach for stack overflow," *arXiv preprint arXiv:2202.09789*, 2022.

[15] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out*, 2004, pp. 74–81.

[16] M. Allahyari, S. Pouriyeh, M. Assefi, S. Safaei, E. D. Trippe, J. B. Gutierrez, and K. Kochut, "Text summarization techniques: a brief survey," *arXiv preprint arXiv:1707.02268*, 2017.

[17] W. S. El-Kassas, C. R. Salama, A. A. Rafea, and H. K. Mohamed, "Automatic text summarization: A comprehensive survey," *Expert Systems with Applications*, vol. 165, p. 113679, 2021.

[18] I. Cachola, K. Lo, A. Cohan, and D. S. Weld, "Tldr: Extreme summarization of scientific documents," *arXiv preprint arXiv:2004.15011*, 2020.

[19] W. Yuan, P. Liu, and G. Neubig, "Can we automate scientific reviewing?" *arXiv preprint arXiv:2102.00176*, 2021.

[20] J. Gu, Z. Lu, H. Li, and V. O. Li, "Incorporating copying mechanism in sequence-to-sequence learning," *arXiv preprint arXiv:1603.06393*, 2016.

[21] R. Paulus, C. Xiong, and R. Socher, "A deep reinforced model for abstractive summarization," *arXiv preprint arXiv:1705.04304*, 2017.

[22] A. Celikyilmaz, A. Bosselut, X. He, and Y. Choi, "Deep communicating agents for abstractive summarization," in *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, Jun. 2018, pp. 1662–1675. [Online]. Available: https://aclanthology.org/N18-1150

[23] D. Bahdanau, K. Cho, and Y. Bengio, "Neural machine translation by jointly learning to align and translate," *arXiv preprint arXiv:1409.0473*, 2014.

[24] A. See, P. J. Liu, and C. D. Manning, "Get to the point: Summarization with pointer-generator networks," *arXiv preprint arXiv:1704.04368*, 2017.

[25] S. J. Rennie, E. Marcheret, Y. Mroueh, J. Ross, and V. Goel, "Self-critical sequence training for image captioning," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7008–7024.

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

[27] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[28] Q. Lhoest, A. Villanova del Moral, Y. Jernite, A. Thakur, P. von Platen, S. Patil, J. Chaumond, M. Drame, J. Plu, L. Tunstall, J. Davison, M. Šaško, G. Chhablani, B. Malik, S. Brandeis, T. Le Scao, V. Sanh, C. Xu, N. Patry, A. McMillan-Major, P. Schmid, S. Gugger, C. Delangue, T. Matussière, L. Debut, S. Bekman, P. Cistac, T. Goehringer, V. Mustar, F. Lagunas, A. Rush, and T. Wolf, "Datasets: A community library for natural language processing," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online and Punta Cana, Dominican Republic: Association for Computational Linguistics, Nov. 2021, pp. 175–184. [Online]. Available: https://aclanthology.org/2021.emnlp-demo.21

[29] C. van der Lee, A. Gatt, E. van Miltenburg, and E. Krahmer, "Human evaluation of automatically generated text: Current trends and best practice guidelines," *Computer Speech & Language*, vol. 67, p. 101151, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S088523082030084X

[30] Y. Guo, W. Qiu, Y. Wang, and T. Cohen, "Automated lay language summarization of biomedical scientific reviews," *arXiv preprint arXiv:2012.12573*, 2020.

[31] A. Bhattacharjee, S. S. Nath, S. Zhou, D. Chakroborti, B. Roy, C. K. Roy, and K. Schneider, "An exploratory study to find motives behind cross-platform forks from software heritage dataset," in *Proceedings of the 17th International Conference on Mining Software Repositories*, 2020, pp. 11–15.

[32] Y. Yu, H. Wang, G. Yin, and T. Wang, "Reviewer recommendation for pull-requests in github: What can we learn from code review and bug assignment?" *Information and Software Technology*, vol. 74, pp. 204–218, 2016.

[33] J. Jiang, Q. Wu, J. Cao, X. Xia, and L. Zhang, "Recommending tags for pull requests in github," *Information and Software Technology*, vol. 129, p. 106394, 2021.

[34] L. Moreno, G. Bavota, M. Di Penta, R. Oliveto, A. Marcus, and G. Canfora, "Automatic generation of release notes," in *Proceedings of the 22nd acm sigsoft international symposium on foundations of software engineering*, 2014, pp. 484–495.

[35] X. Li, H. Jiang, D. Liu, Z. Ren, and G. Li, "Unsupervised deep bug report summarization," in *2018 IEEE/ACM 26th International Conference on Program Comprehension (ICPC)*. IEEE, 2018, pp. 144–14 411.

[36] S. Xu, Y. Yao, F. Xu, T. Gu, H. Tong, and J. Lu, "Commit message generation for source code changes," in *IJCAI*, 2019.

[37] D. Q. Nguyen, T. Vu, and A. T. Nguyen, "Bertweet: A pre-trained language model for english tweets," *arXiv preprint arXiv:2005.10200*, 2020.