electronics

MDPI

Article

A Comparison of Summarization Methods for Duplicate Software Bug Reports

Samal Mukhtar [1,†], Claudia Cahya Primadani [1,†], Seonah Lee [1,2,*] and Pilsu Jung [1,*]

[1] Department of AI Convergence Engineering, Gyeongsang National University, 501 Jinjudaero, Jinju-si 52828, Gyeongsangnam-do, Republic of Korea; aurelisea@gmail.com (S.M.); cc.primadani@gmail.com (C.C.P.)
[2] Department of Aerospace and Software Engineering, Gyeongsang National University, 501 Jinjudaero, Jinju-si 52828, Gyeongsangnam-do, Republic of Korea
* Correspondence: saleese@gnu.ac.kr (S.L.); psjung@gnu.ac.kr (P.J.); Tel.: +82-55-772-1377 (S.L.); +82-55-772-1372 (P.J.)
† These authors contributed equally to this work.

Abstract: Bug reports vary in length, while some bug reports are lengthy, others are too brief to describe bugs in detail. In such a case, duplicate bug reports can serve as valuable resources for enriching bug descriptions. However, existing bug summarization methods mainly focused on summarizing a single bug report. In this paper, we focus on summarizing duplicate bug reports. By doing so, we aim to obtain an informative summary of bug reports while reducing redundant sentences in the summary. We apply several text summarization methods to duplicate bug reports. We then compare summarization results generated by different summarization methods and identify the most effective method for summarizing duplicate bug reports. Our comparative experiment reveals that the extractive multi-document method based on TF-IDF is the most effective in the summarization. This method successfully captures the relevant information from duplicate bug reports, resulting in comprehensive summaries. These results contribute to the advancement of bug summarization techniques, especially in summarizing duplicate bug reports.

Keywords: bug summarization; duplicate bug reports; comparative experiment; multi-document summarization; software maintenance; bug management system

Citation: Mukhtar, S.; Primadani, C.C.; Lee, S.; Jung, P. A Comparison of Summarization Methods for Duplicate Software Bug Reports. *Electronics* **2023**, *12*, 3456. https://doi.org/10.3390/electronics12163456

Academic Editor: Andrei Kelarev

Received: 18 July 2023
Revised: 8 August 2023
Accepted: 10 August 2023
Published: 15 August 2023

1. Introduction

Software bugs are inevitable in software development. Bug reports play a crucial role in reporting bugs and tracking the progress of fixing bugs. Furthermore, reporters and developers often hold lengthy discussions to understand bugs in bug reports. With all of these activities, bug reports become lengthy, but not every sentence in a bug report is valuable.

To provide developers with a brief bug description, researchers developed bug summarization techniques. However, they mainly focused on summarizing a single bug report, while a bug report can be long with a lot of information, such as code snippets, error logs, reproductions steps, and links [1], a certain bug report is very brief without sufficient details. In this case, duplicate bug reports may contain additional information that is useful for performing bug fixing tasks [2,3]. However, previous bug summarization techniques did not provide a comprehensive bug summary by utilizing duplicate bug reports.

In our view, it is important to summarize the bug report with valuable information, and duplicate bug reports can be affluent resources to provide informative bug summaries. Therefore, we explore different methods for summarizing duplicate bug reports. We identify six methods to apply to the task of summarizing duplicate bug reports. First, we adopted multi-document summarization methods for bug reports (KCMS [4] and MDNS [5]). Second, we modified existing two single-document summarization methods

(T5 [6] and BART [7]). Third, we used two the state-of-the-art methods (PEGASUS [8] and PRIMERA [9]).

To evaluate these models, we collected 16 pairs of bug reports from the VSCode repository on GitHub as a dataset. Two annotators created a golden summary for each pair of bug reports. Based on the golden summaries, we evaluated the outcomes from each summarization methods. As a result, we found that the summarization method MDNS outperforms the six other summarization methods. The MDNS method is an extractive multi-document summarization method that puts weights on important sentences, considering the redundancy of sentences.

The contributions of our research are as follows:

1. We apply and evaluate several text summarization methods for duplicate bug reports;
2. We compare and analyze the effectiveness of existing text summarization methods for duplicate bug report summarization;
3. We analyze the impact of using duplicate bug reports for bug summarization.

The rest of this paper is organized as follows. Section 2 discusses related works. Section 3 presents the six summarization methods used for comparison. Section 4 offers the material and experiment procedure, followed by results in Section 5. Section 6 discusses implications and threats to validity. Section 7 concludes the paper.

2. Related Work

There are two types of summarization methods: extractive and abstractive summarization [4]. Extractive summarization generates a summary by selecting sentences and listing them in the same order [1,10]. Abstractive summarization generates a summary by selecting a representative sentence, combining and then modifying the sentence into a general sentence with the same contextual meaning [1,4]. Section 2.1 introduces extractive summarization methods for bug reports. Section 2.2 introduces abstractive summarization methods for bug reports. Section 2.3 introduces summarization methods with duplicate bug reports.

2.1. Extractive Summarization for Bug Reports

We found that researchers have developed extractive summarization methods for bug reports from 2012 to 2021 [11–18]. When the types of machine learning are considered, the summarization methods can also be divided into supervised learning and unsupervised learning. Supervised learning is an approach for training a model with tagged data. Unsupervised summarization refers to the method of summarizing a text or document without using a trained model.

2.1.1. Extractive Supervised Summarization for Bug Reports

An early research phase of summarizing bug reports focused on extractive supervised summarization methods [12,15]. Rastkar et al. proposed automatic summarization based on existing conversation-based summarization. They used and published their datasets or corpus containing manual summary references. Their approach uses logistic regression and is trained on their corpus (BRC) to produce a summary. They conducted task-evaluation-based to validate whether bug summaries help developer performance in fixing bugs [12].

Yang et al. proposed a two-layer semantic model (TSM) extending the previous semantic model (AUSUM). Their work implemented noise filtering (NR) and proposed two more classes (anthropogenic and procedural) to identify informative or essential information on the first layer. Then, they trained the summarizer based on BRC feature extraction by Rastkar in the second layer [15].

Those two research groups focused on a single bug report because they found that bug reports usually contain long conversations between the reporter and developer. They proposed summarizing a single bug report to shorten the long discussion in a bug report.

2.1.2. Extractive Unsupervised Summarization for Bug Reports

Unsupervised summarization methods have the obvious advantage of not requiring any labeled data. Therefore, many researchers focused on summarizing bug reports with unsupervised summarization methods [11,13,14,16–19].

Mani et al. evaluated four unsupervised techniques and presented an AUSUM that use semantic information to summarize bug reports on SDS datasets and one industrial project dataset (DB2-Bind). They also applied noise identifiers and filtering, which improved the precision compared to unsupervised base techniques, and classified the sentence into four semantic classes: question, code, investigation, and others. The study identified and analyzed the efficacy of techniques and compared them with supervised techniques. The threat of implementing this model is the possibility of removing essential information outside the identified classes because of a noise reducer [11].

Latufo et al. created an automatic unsupervised bug report summarization that ranks and selects the sentences most relevant to titles and descriptions of bug reports. Their work focused on finding the essential parts of bug reports and developing a reliable model. For the evaluation, they used the BRC corpus by Rastkar. They also validated their work by letting developers assess the quality and usefulness of bug report summaries [13].

Ferreira et al. proposed a bug report summarization by ranking comments using four techniques. Their strategy was to rank comments in the order of the most relevant to the bug report description. Their strategy helped obtain more accurate and appropriate information than a sentence-based ranking strategy. They evaluated their work by creating manual bug report summaries. Their extractive summarization technique enabled the developers who used bug summaries produced by the proposed method to obtain more appropriate information [14].

Liu et al. introduced BugSum, an unsupervised bug summarization approach that integrates an auto-encoder network for feature extraction. BugSum produced the summary by extracting sentence features and used a novel metric, believability, to measure the degree of a sentence. To evaluate BugSum, the authors used two data sets of Authorship Dataset (ADS) and Summary Dataset (SDS). They reported that the method prevented controversial sentences from being included in the produced summary [16].

Kukkar et al. applied an unsupervised method using swarm intelligence to generate an extractive summary of bug reports. The particle swarm optimization subset selection and ant colony optimization subset selection is the method with the application of Kullback—Leibler divergence to the summarization problem, which helps to choose the best subset of summaries generated according to feature scoring. The method could be applied to any dataset without a manual summary. They evaluated their study using 36 bug reports from the Ratskar dataset [17].

Sastri et al. presented the Multi-objective Evolutionary Algorithm for Bug Report Summarization (MEABRS), an automatic unsupervised bug report summarization technique implemented without annotated data. Their work also implemented the Rapid Automatic Keyword Extraction (RAKE) toolkit to improve the quality of bug report summaries. They evaluated their extractive summarization model on two popular benchmark datasets, ADS and SDS [18].

Koh et al. developed a deep-learning-based bug report summarization method to overcome limitations in the existing approach [15]. They introduced the concept of sentence significance factors, which are criteria for identifying important sentences. Their method outperformed the state-of-the-art BugSum method in terms of precision, recall, and F score. The contributions of their work include proposing a comprehensive summarization method that considers multiple sentence significance factors and demonstrating its superiority over BugSum using two benchmark datasets [19].

They also focused on summarizing a single bug report, aiming to reduce the length of a bug report. Our study differs from theirs in Sections 2.1.1 and 2.1.2, in that we focus on summarizing duplicate bug reports as multiple documents.

2.2. Abstractive Summarization for Bug Reports

Abstractive summarization techniques have the ability to generate new sentences and phrases that capture the essence of the source text. There are a few approaches on abstractive summarization for bug reports [20,21].

Huai et al. proposed an intention-based bug report summarization. They classified the bug report sentences into seven categories of intention, since the reader of the bug report is usually more interested and skims the content of the bug report based on specific categories such as description, fixing solution, and meta/code. They evaluated their approach by extending the corpora published by Ratskar BRC into Intention-BRC [20].

Tian et al. applied the Transformer pre-trained model T5 to generate bug report titles and present the core idea of bug reports. The method used SentencePiece to tokenize and detokenize the sentence to fine-tune the original T5 model onward, called BUG-T5 [21].

Ma et al. proposed a deep attention-based summarization model called ATTSUM to generate high-quality bug report titles. It utilizes a bidirectional-encoder-representations-from-transformers approach to capture contextual semantic information, a stacked transformer decoder to generate titles, and a copy mechanism to handle rare tokens. The effectiveness of ATTSUM is validated through automatic and manual evaluations on a large dataset of bug reports. Results show that ATTSUM outperforms existing baselines in terms of evaluation metrics and human evaluations. Additionally, the impact of training data size on ATTSUM is analyzed, indicating its robustness in generating improved titles [22].

While abstracted summaries have the advantage of being more concise, they tend to convey only general information without specific details. Furthermore, the studies above are more specialized in creating titles for bug reports rather than summarizing their contents. Our study differs from the existing studies in that we focus on extracting essential specific contents from duplicate bug reports.

2.3. Summarization with Duplicate Bug Reports

Similarly to ours, there are previous studies that made efforts to utilize information from duplicate bug reports [23,24].

Jiang et al. proposed the PageRank-based summarization (PRST) method for generating summaries from a bug report with additional information in associated duplicate bug reports by applying three variants of the PageRank-based algorithm. The method identifies and calculates the similarity of textual information between sentences in an original bug report and their duplicates using three similarity metrics. The study used the Reconstruction of Bug Report Corpus (MBRC) and constructed a new corpus called the OSCAR corpus [23].

Kim et al. introduced an unsupervised bug report summarization method called the relation-based bug report summarizer (RBRS). This approach utilizes dependency relationships and duplicate associations between bug reports. RBRS creates a bug report graph with relevant bug reports based on these relationships, and then applies the weighted PageRank algorithm to assign a score to each sentence in the bug report. While similar to another approach proposed by [23], RBRS does not include a regression module, but instead focuses on bug report features and considers relations between documents [24].

Their approaches mainly extended the summarization technique of a single bug report. Our approach differs from theirs in that we conduct a comparative study to find a summarization technique to describe more complete information from duplicate bug report, including multi-document summarization techniques [10] (multi-document summarization can take two or more documents as inputs, thus making summaries more complete with additional information).

3. Six Summarization Methods for Duplicate Bug Reports

We identified six summarization models to summarize duplicate bug reports. In this research, we were interested in applying multi-documentation summarization techniques to summarize duplicate bug reports. However, we did not know if the performance of

techniques for summarizing a single bug report would differ if the techniques were applied to summarizing duplicate bug reports. Furthermore, there were possibilities that state-of-the-art techniques may perform better on the same problem than the multi-documentation summarization techniques we are interested in. Therefore, we selected two representative summarization techniques from each of the three categories.

As there were no previous studies that applied a multi-document summarization technique to bug reports, we first searched general multi-document summarization techniques and altered the inputs to bug reports. The first model we chose combines four different techniques to overcome overlapping information problems, and the second one is based on terms of importance identification, which is valuable in domain-specific tasks like bug report summarization. We named them KCMS and MDNS, respectively. We next chose deep learning models that give promising results in text summarization, reaching state-of-the-art results, and modified them to summarize each 512-sized section for performance improvement, not only the first one. We selected two state-of-the-art summarization models for multi- and single-document summarization, respectively.

Table 1 shows the six summarization models. Two models, KCMS and MDNS, are characterized by multi-document summarization. Two models, T5 and Bart, are characterized by deep learning models for single documents. Two models, PEGASUS and PRIMERA, are characterized by the state-of-the art techniques. Sections 3.1–3.6 explain these models.

Table 1. The Characteristics of Six Summarization Models.

Group	Model Name	Type 1	Type 2
Alteration for Bug Reports	KCMS	Extractive	Multi-documents
Alteration for Bug Reports	MDNS	Extractive	Multi-documents
Modified Version	T5	Abstractive	Single-documents
Modified Version	BART	Abstractive	Single-documents
State-of-the-art Model	PEGASUS	Abstractive	Single-documents
State-of-the-art Model	PRIMERA	Abstractive	Multi-documents

3.1. K-Means, Centroid-Based, MMR, and Sentence Position (KCMS)

The first approach, KCMS, is an extractive multi-document summarization method proposed by Manh et al. [4]. The challenge of implementing multi-document summarization is that the information generated in a summary can become redundant, since there is overlapping information among salient sentences. Manh et al. overcame the problem by implementing a k-means clustering algorithm combined with the centroid-based method, maximal marginal relevance, and sentence positions. Figure 1 represents the architecture of the model. The multi-document summarization model consists of two main modules: input processing and extractive summarization. Sections 3.1.1 and 3.1.2 explain each module, respectively.

3.1.1. Input Processing

The first module, input processing, process the bug report by removing stop words, stemming the input words, and converting each input sentence into a vector. Manh et al. used the bag of words (BoW) model as a vector representation. However, the BoW model does not contain the semantic meaning of a sentence. Therefore, we altered the vector representation to Google's pre-trained Word2vec to improve the performance. As a result, the module generates the embedding vectors used as input for the second module.

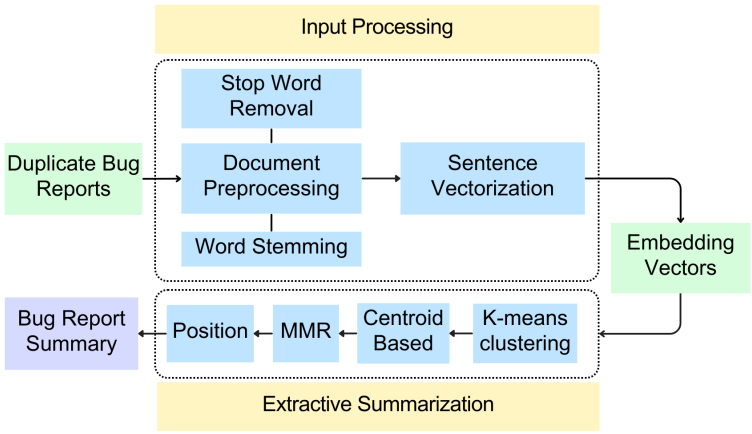

Figure 1. Architecture of extractive multi-document summarization using KCMS.

3.1.2. Extractive Summarization

The second module, extractive summarization, generates a summary of duplicate bug reports from the vector of sentences by extracting the most informative or essential sentences. In this module, Manh et al. used the k-means clustering algorithm at the beginning of the process. The k-means algorithm groups the vector of sentences into clusters. Those clustered sentences are candidate sentences for further summarization. However, not all sentences in a cluster are appropriate or necessary. Therefore, we applied a centroid-based method to select the most salient sentence in a cluster. Now, we obtain the most salient sentences from clusters. The challenge of implementing multi-document summarization is that the information is possibly redundant. Therefore, this module implements the Maximal Marginal Relevance (MMR) to remove redundant sentences. Lastly, the module orders the selected sentence using the information on sentence position and generates a summary of duplicate bug reports.

3.2. Multi-Doc News Summarizer (MDNS)

The second approach, MDNS, is an extractive unsupervised multi-document summarization method that extracts representative sentences to generate a summary [5]. The model uses TF-IDF word frequency and the relative sentence location in documents to generate the summary. Figure 2 represents the architecture of the model. This MDNS approach regards the sentences at the beginning and ending positions of a bug reports as the most important sentences. Furthermore, the approach considers that a sentence with a long length will be more valuable than a sentence with a short length. The model consists of two modules: text preprocessing and score assignment. Sections 3.2.1 and 3.2.2 explain each module, respectively.

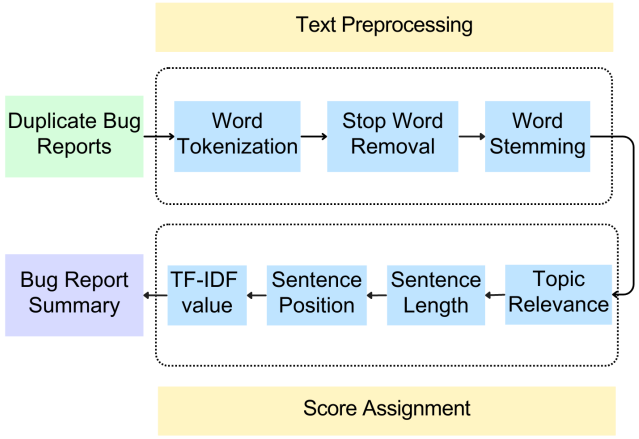

Figure 2. Architecture of extractive multi-document summarization using MDNS.

3.2.1. Text Preprocessing

The first module, text preprocessing, includes word tokenization and word stemming. This model uses the standard NLTK library for tokenization, the stop-word corpus, and word stemming.

3.2.2. Score Assignment

The second module, score assignment, assigns scores to each sentence extracted from the first module based on the weights of text features. The feature weights are calculated in the subsequent phases. First, the module starts by looking for words that are relevant to the topic, subject, or headline. Second, the module measures the length of a sentence. Third, the module identifies where the relevant sentence is located in an input bug report. Finally, the module estimates the word frequency value using the TF-IDF. The module selects the sentences with the highest feature weights for generating the final summary.

3.3. T5

T5 (Text-To-Text Transfer Transformer) is a transformer-based model introduced by Google AI for text generation or summarization [6]. However, T5 has a limited embedding length, only 512. Due to the limited length, T5 only generates a brief summary with 10–15% of the original bug report. Meanwhile, for an optimal summary, the length of the summary generated should be around 25% of the original bug report [24].

Considering the limitation on the embedding length of T5, we decided to modify the T5 model by adding a part for dividing the bug report input into sections so that the T5 model can generate a more lengthy and meaningful summary. However, the length of a bug report varies, and each bug report has no fixed, standard amount of words. Therefore, we decided to divide the bug report into sections based on tokens. Moreover, such a division method helps to manage the length by adding the content from duplicate files. Figure 3 describes the T5 method we used for duplicate bug report summarization.

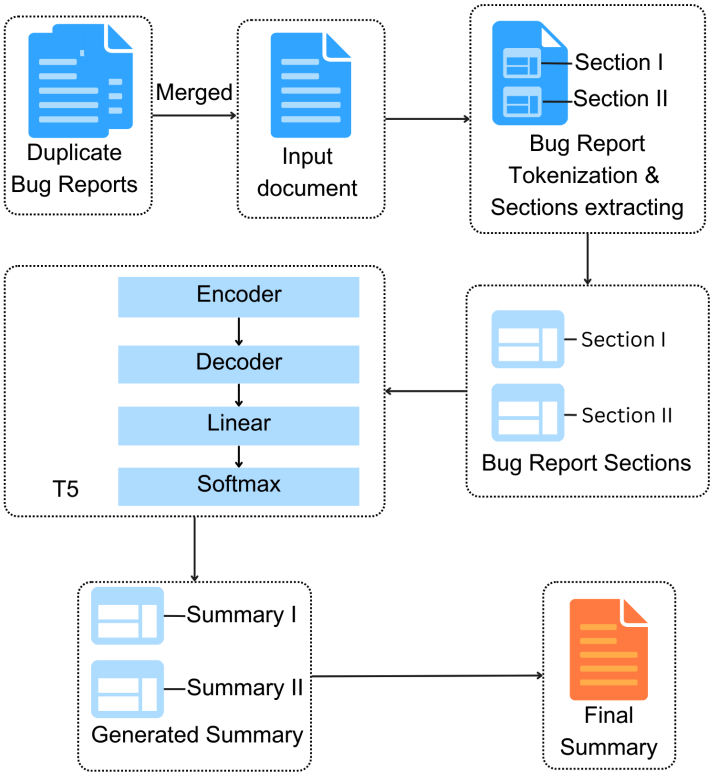

Figure 3. Architecture of the T5 model for summary generation.

In the first step, the T5 model reads the entire contents of bug reports as input and then tokenizes the inputted text. If the tokenized input exceeds the limit of embedding length of 512, the model slices the input as a section, stores the section into the section list, and repeats the process for the remaining part of the bug report. As a result, we have lists of bug report sections that maintain tokenized text input for the next step.

In the second step, the T5 model generates a summary from the section list of tokenized text. We used the default configuration of T5 models and set the number of beams to 4. The model generates a summary from each section of the section list. The model merges the summary list of each section into the final summary. With this approach, we can obtain the optimal 25% of the length of the original bug report.

3.4. BART

Bidirectional and Auto-Regressive Transformers (BART) is a sequence-to-sequence model architecture combining auto-regressive and denoising objectives for pre-training [7]. Facebook AI introduced BART and evaluated it with one of the benchmark and analysis platforms, namely GLUE [25].

Similar to other sequence-to-sequence text summarization models such as T5, BART also has limitations in terms of context length. Therefore, to prevent the limitation of the context length, we modified the model to split the bug report input into sections. Figure 4 shows that T5 and BART are interchangeable in generating a summary. Overall, we utilize the BART model in a similar way we did the T5.

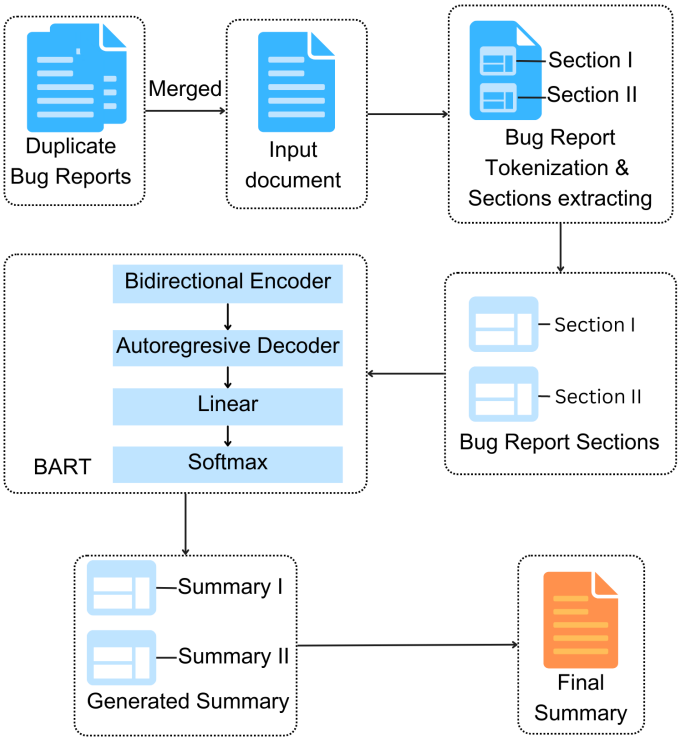

Figure 4. Architecture of the BART model for summary generation.

3.5. PEGASUS

PEGASUS is an abstractive sequence-to-sequence summarization model with an encoder–decoder architecture similar to BART [8], described in Figure 5. The model uses an encoder–decoder architecture, with the encoder processing the input document and the decoder generating the summary. The extractive gap-filling during pre-training involves masking entire sentences with special tokens, and the model is trained to predict these missing sentences. The target tokens during training are the original sentences that were removed, and the model is trained to generate summaries that align with these targets.

Zhang et al. evaluated the performance of PEGASUS with 12 downstream tasks. We used the model with the default configuration settings, except for the number of beams.

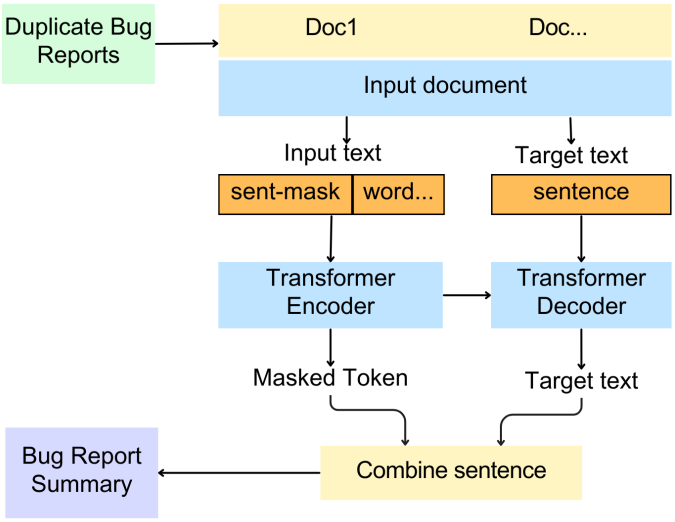

Figure 5. Architecture of the PEGASUS model for summary generation.

Since the other pre-trained models (T5 and BART) use the same number of beams in our experiment, we set up the same number for the PEGASUS to compare its performance with those of other models.

3.6. PRIMERA

PRIMERA is an abstractive multi-document summarization model pre-trained with multiple large datasets. PRIMERA focuses on reducing the dependency on the specific dataset and labeled data [9]. The model utilizes a hierarchical encoder–decoder architecture, with the encoder processing the input documents at different levels of granularity and the decoder generating the summary. The masked tokens during pre-training involve replacing segments at different levels with special tokens, and the model is trained to predict these missing segments. The target tokens during training are the original segments that were masked, and the model is trained to generate summaries that align with these targets, considering the hierarchical relationships within and between the input documents. The authors explore the effectiveness of PRIMERA by comparing it with multiple existing pre-trained models. They conducted an additional human evaluation to show its capability to generate more fluent summaries and capture salient sentences. Figure 6 describes the architecture of PRIMERA.

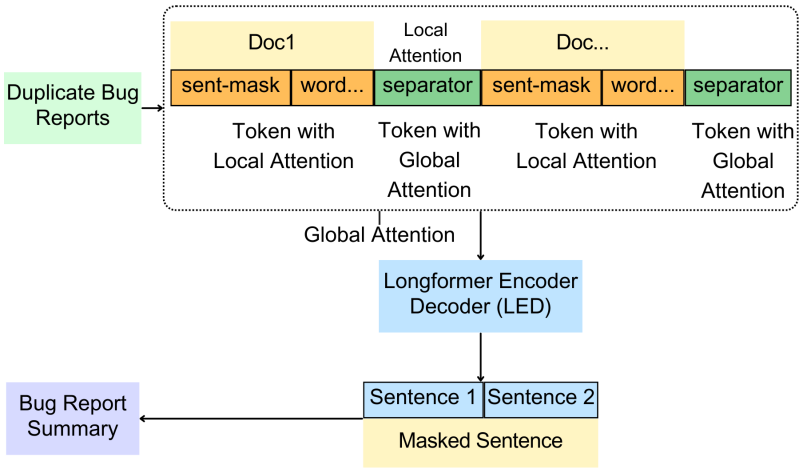

Figure 6. Architecture of the PRIMERA model for summary generation.

We utilized PRIMERA from the HuggingFace hub and implemented it with the default configuration settings. The only parameter we changed is the number of beams. We set it to 4, the same number of beams that other state-of-the-art models maintain. We made this change to compare its performance with those of other models in a fair condition.

4. Experimental Setup

Section 4.1 identifies the research questions. Section 4.2 identifies the datasets. Section 4.3 explains the data preprocessing, and Section 4.4 explains the annotation process of the bug reports. Section 4.5 identifies the evaluation metrics for comparing the performance of several summarization models.

4.1. Research Questions

We formulated two research questions to evaluate and analyze the performance of the proposed approach for bug report summarization. Our experiments answer the following questions:

- **RQ1**—Which approach yields the highest performance for duplicate bug report summarization?
- **RQ2**—How well does each approach comprehensively summarize the key information in bug reports?

To answer RQ1, we quantitatively compared the performance values of the six summarization models described in Section 3. To answer RQ2, we qualitatively investigated the summarization results of each model.

4.2. Datasets

To assess the performance of the six summarization models, we collected 16 duplicate bug reports from the VSCode repository on the GitHub platform. The collection was performed manually in three steps. First, we randomly selected bug reports from the VSCode repository. Second, when selecting bug reports, we only considered the reports with the labels "bug" and "duplicate". We also investigated whether the bug report had a minimum of one duplicate report. We extracted content from chosen bug reports and their duplicates. Third, we checked if the bug report contained interesting content. We considered such attributes as descriptions of both original bug reports, their duplicates, and their discussion content in the comment section. We organized extracted content putting descriptions and comments of the original bug report and the same attributes of its duplicated reports in separate textual files. Table 2 shows 16 duplicate bug reports, where 14 of them have one duplicate report and two contain two or three duplicated reports.

4.3. Data Preprocessing

The dataset used in this study consists of duplicate bug reports as shown in Table 2. We regarded the duplicate bug reports as multiple documents. In our experiment, we employed not only multi-document summarization models but also single-document summarization models. Therefore, for the single-document model, we merged the duplicate reports into a single document for each dataset. The main reason for this action is to ensure that the single-document model also gets the full context of the bug report, as the multi-document model gets duplicate bug reports to generate a summary.

When it comes to the data from extracted bug reports, we only used the descriptions and discussions recorded in them because we summarized the textual contents. In practice, bug reports can contain descriptions, code snippets, logs, external link references, and discussions between the reporter and developer. In this study, we removed snippet code, logs, and external links.

Table 2. Sets of duplicate bug reports from the VSCode GitHub repository.

ID	Title	Number of Duplicates	Issue Numbers
1	Git Bash is not visible when try to select default terminal profile	2	#126023, #158627
2	VS Code was lost on shutdown with pending update	2	#52855, #161019
3	Loses text when maximizing the integrated terminal	2	#134448, #159703
4	The terminal cannot be opened when multiple users use remote-ssh	2	#157611, #159519
5	GitHub Commit error after June VS Code update	4	#154449, #154463, #154504, #154837
6	Ansi Foreground colors not applied correctly when background color set in terminal	2	#155856, #146168
7	View locations reset	2	#156315, #154090
8	Webview Panels Dispose Bug	2	#158839, #98603
9	Open file and then search opens the wrong find widget	2	#156853, #155924
10	Sticky Scroll: Go to definition jumps to definitions but stay hidden behind sticky scroll layout	2	#157225, #157175
11	bash: printf: 'C': invalid format character error in bash terminal	2	#157278, #157226
12	New rebase conflicts resolution completely broken. Diff worse than before.	3	#157735, #156608, #157827
13	Underscores (_) in integrated terminal are surrounded by black box	2	#158522, #158497
14	Searching keyboard shortcuts with 'Record Keys' does not include chords using that keybinding	2	#158799, #88122
15	Trust dialog prompt appears even when disabled in settings	2	#159823, #156183
16	Monolithic structure, multiple project settings	2	#32693, #155017

4.4. Annotation of Bug Reports

Since the annotation process is crucial for evaluating summarization tasks, we manually annotate each bug report of a corpus. To create the golden truth, two of the first authors in this paper participated in the annotation tasks. The two authors have at least five years of experience in programming and research experience in issue reports. Each author annotated each bug report and made a consolidation for the conflicts arising between the two authors' decisions. By making a consolidation, we intended to reduce the subjectivity and bias from one human's decision. Each selected sentence from the original and duplicate bug reports is regarded as the most informative and important, so it becomes the sentences in the golden summaries.

4.5. Evaluation Metrics

We used the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) toolkit as a performance metric to evaluate and compare all proposed models for summarizing duplicate bug reports. ROUGE measures the quality of the generated summary by counting continuously overlapping units between the generated summary and the ground truth.

$$Rouge\text{-}n = \frac{\sum_{s \epsilon GT} \sum_{n\text{-}grames} Count_{match}(n\text{-}gram)}{\sum_{s \epsilon GT\text{-}grames} Count(n\text{-}gram)} \quad (1)$$

Formula (1) describes the ROUGE metric we used to evaluate the generated summary. Here, *s* refers to the generated sentence, *GT* refers to the reference summary, and *n-gram* is a contiguous sequence of *n* items as words and characters within a sentence. The numerator counts the number of n-grams between the generated and golden summaries. The denominator counts the number of the *n-grams* in the golden summary. We set up *n-gram* lengths as 1 and 2.

In addition, we also compared the result with standard evaluation metrics Precision, Recall, and F1 score, to measure the accuracy of the generated bug report summary [11,12]. Precision measures the proportion of relevant content in the generated summary compared to the total content in the generated summary. In comparison, Recall measures the proportion of relevant content in the generated summary compared to the total relevant content in the reference summary. The F1 score is a harmonic mean of precision and recall. It provides a balanced evaluation of both precision and recall.

5. Results

In this study, we asked one quantitative research question (RQ1) and one qualitative research question (RQ2). Section 5.1 discusses the experimental results for RQ1 and Section 5.2 discusses the experimental results for RQ2.

5.1. Research Question 1

To answer RQ1, we investigated the performance of the six models that we presented in Section 3 in terms of the ROUGE values. Figure 7 presents the experimental results, demonstrating the ROUGE performance values of different approaches.

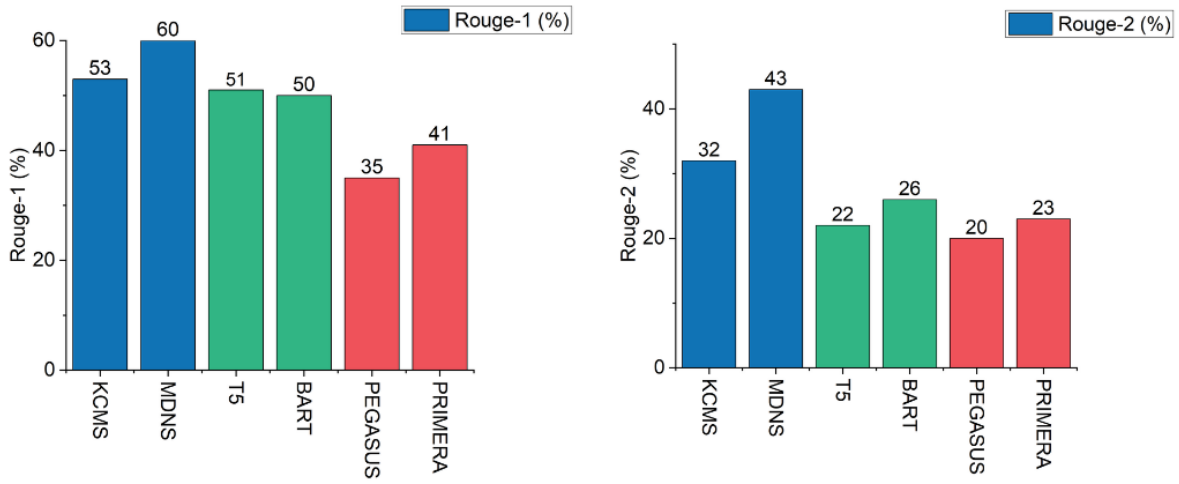

Figure 7. ROUGE-n results of summarization models.

According to the experiment result, the second model MDNS, which uses TF-IDF, outperforms the other models. The MDNS achieved 60% and 43% for ROUGE-1 and ROUGE-2 scores, respectively. It can be interpreted as that MDNS is more effective in generating summaries, considering that it achieved the highest performance in our quantitative evaluation of RQ1.

The two extractive summarization methods, KCMS and MDNS, produced promising outcomes. Meanwhile, the two abstractive summarization methods, T5 and BART, produced inferior results compared to KCMS and MDNS. The results are as expected and in line with the previous studies, which stated that the abstractive summarization model requires more contextual information and complex procedure to produce a good summary [1,10].

Nevertheless, the results of BART and T5 that we propose significantly outperform the other two state-of-the-art models that we utilized for comparison, PEGASUS and PRIMERA.

PEGASUS and PRIMERA were not initially trained on bug report data. PEGASUS is trained and evaluated using news data [8], while PRIMERA is trained using a variety of data sources, including news, DUC2003/2004, and Wikisum [9].

However, PEGASUS and PRIMERA performed less effectively than the first four models in summarizing duplicate bug reports. This leads to the conclusion that these models, which achieved state-of-the-art performance in one case study, do not necessarily show high performance in other case studies overall.

We re-evaluated the six models with a different measure, the F1 score. Figure 8 shows the F1 scores of the six models. As shown in Figure 8, MDNS has the highest F1 score (41%) among all models.

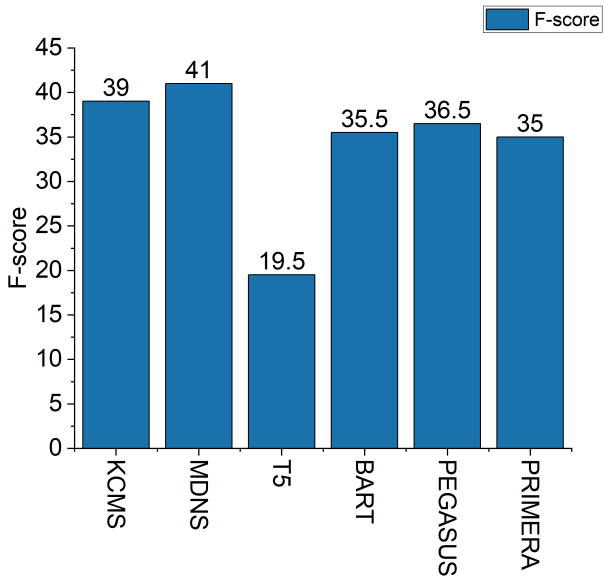

Figure 8. F1 scores of summarization models.

An F1 score in Figure 8 is an average of the F1 scores of a summarization model, based on the summaries manually created by two referees (i.e., the two first authors of this paper). To further analyze the summarization results, Figures 9 and 10 show the Precision, Recall, and F1 scores of summarization models, based on respective manual summaries of two referees.

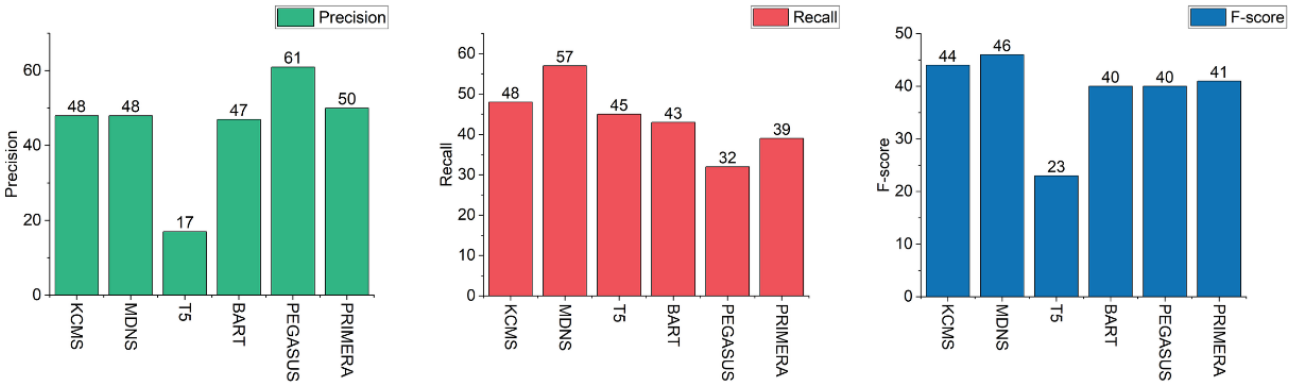

Figure 9. Precision, Recall, and F1 score of the first reference summaries.

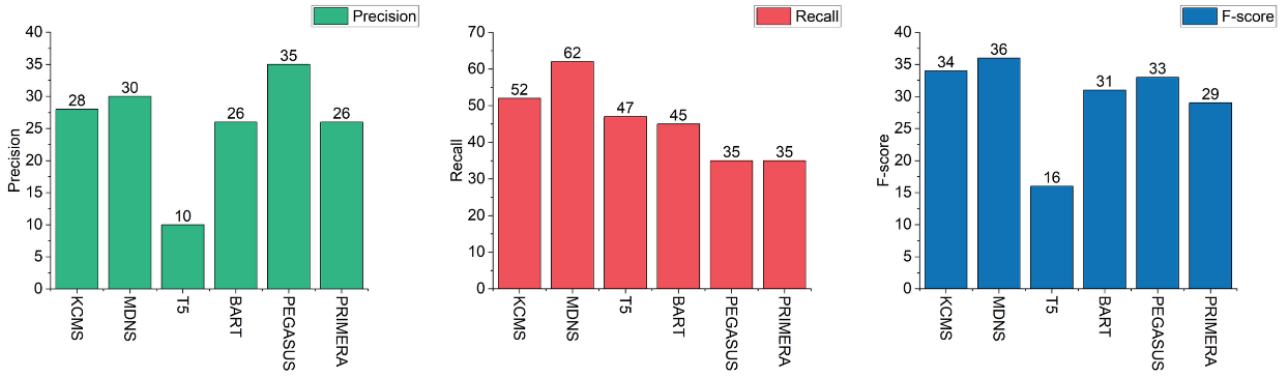

Figure 10. Precision, Recall, and F1 score of the second reference summaries.

The MDNS method shows higher F1 scores than other models. At the same time, the summaries generated by this model are also the longest. That is, the probability of choosing the necessary and unnecessary sentences by the model is higher, as we can see from the Precision and Recall scores in Figures 9 and 10. The high recall scores of 57% and 62% suggest that the summaries generated by the model include a large proportion of the vital information in the reference summaries. The reasonable precision scores of 48% and 30% indicate that the summaries do not contain a significant amount of irrelevant or incorrect information. However, the Precision score is lower than those of the PEGASUS (61% and 35%) and PRIMERA (50% and 26%), which means that those models are more accurate than MDNS. However, the Recall scores of those models are lower than other models. The summaries of those models lose most of the necessary information.

5.2. Research Question 2

To answer RQ2, we investigated the summary results generated by each method (see Appendix A if you would like to check the complete summary results). Table 3 shows the overall results of our investigation. As shown in Table 3, we found that MDNS covers the three elements of a bug report: bug description, reproduction step, and possible solution. Meanwhile, other methods missed at least one of them.

Table 3. The coverage of summary contents.

Model	Bug Description	Reproduction Step	Possible Solution
KCMS	Δ	O	X
MDNS	O	O	O
T5	Δ	X	X
BART	O	X	X
PEGASUS	Δ	X	X
PRIMERA	Δ	X	X

We also found that the length of the summary generated by MDNS is the longest among the summary results generated by other methods. Therefore, we conjecture that the length of a summary could affect the quality and overall content of the summary. Meanwhile, the summary generated by MDNS has no sentences with identical meanings, which is the primary concern when we summarize duplicate bug reports. Therefore, we concluded that MDNS is feasible for duplicate bug report summarization.

From now on, we will discuss the summary generated by each method. If we discuss all three elements, bug description, reproduction step, and possible solution, there should be a lengthy discussion with long examples. Therefore, we only focus on the five lines of each summary result, and we mainly investigate if the first five lines of the summary include the bug description.

To qualitatively investigate the summary results of each model, we used a golden summary that we summarized for the bug report #154090 of the VS Code project. Table 4

shows the first part of the golden summary. The bug report addressed a problem with view locations. In both original and duplicate reports, users provided reproduction steps and their opinions on the cause of the issue. Although there were some fixing suggestions, the annotators did not include them in the summary, since it was controversial with another user.

Table 4. Example of a golden summary.

Views location is not getting updated when switching profiles. I think this happened because I was testing profiles. My view locations have been reset. Found an easy repro: • Have 2 windows open (not sure if this step is needed) • Disable profiles in settings (not sure if this step is needed) • Restart. • Enable profiles in settings • Restart In one of the windows run "Settings Profiles: Create an Empty Settings Profile" and create a new empty profile. See that your new profile gets used in just that window. In that same window run "Settings Profiles: Switch" and choose your default profile. See that your default profile is applied, but view locations are reset . . .

We then investigated the summarized content in each of the first five lines of the summary. We checked if the first five lines contained the bug descriptions or not. First, KCMS summarized the bug report with the five lines, as shown in Table 5. The summarized text indicates the problem with view locations when a user opens a profile. However, the summarized text misses that view locations must be updated correctly.

Table 5. An exampled summary generated by KCMS.

Enable profiles in settings Restart In one of the windows run "Settings Profiles: Create an Empty Settings Profile" and create a new empty profile See that your new profile gets used in just that window In that same window run "Settings Profiles: Switch" and choose your default profile See that your default profile is applied, but view locations are reset . . .

We also investigated the text summarized by MDNS, as shown in Table 6. We found that the summarization includes the sentence indicating the main message of the bug report, "Views location is not getting updated when switching profiles". The starting sentence is very understandable. However, the following sentences that describe the reproduction steps of the bug are not as proper as those of the golden summary shown in Table 4. Therefore, there is still room to improve the generated summary.

Table 6. An exampled summary generated by MDNS.

Views location is not getting updated when switching profiles Enable Settings Profiles Create an empty profile P1 and move Timeline and Outline views to the secondary sidebar as separate view containers Create an empty profile P2 Switch from P2 to P1 . . .

We went further into T5 summary results. The text summary just mentions the potential event that may have caused the issue, "I think this happened because I was testing profiles", as shown in Table 7. The summary does not include the primary problems of the bug report. Furthermore, the summary repeats one of the reproduction steps "restart" twenty-two times instead of including the complete reproduction steps.

Table 7. An exampled summary generated by T5.

I think this happened because I was testing profiles. Restart. Restart. Restart. Restart. . . .

We also examined the summary generated by BART, as described in Table 8. We discovered that the summary includes the sentence indicating crucial issues of the bugs, "I took a look at the steps in the/nViews location not getting updated when switching profiles". However, the sentences are unorganized. Besides, the summary contains the newline character "/n" rather than just generating the following phrases with a new line.

Table 8. An exampled summary generated by BART.

View locations reset I think this happened because I was testing profiles. I have Gitlens installed, and/nEnable profiles in settings. Restart. Create a new empty profile./nI think you might be hitting this—#154090. I took a look at the steps in the/nViews location is not getting updated when switching profiles . . .

Meanwhile, when we investigated the summary generated by PEGASUS, we found the interesting fact that while the summary lists the reproduction steps in order, while capturing the behavior of reproducing the problem, it is quite difficult to capture the main problem of the issue from the summary. Furthermore, the summary lacks punctuation symbols. The summary contains seven phrases with just one punctuation symbol, as shown in Table 9.

Table 9. An exampled summary generated by PEGASUS.

Enable profiles in settings Restart In one of the windows run "Settings Profiles: Create an Empty Settings Profile" and create a new empty profile See that your new profile gets used in just that window In that same window run "Settings Profiles: Switch" and choose your default profile See that your default profile is applied, but view locations are reset/duplicate Thanks for the steps . . . profiles . . .

Last, we investigated the summary generated by PRIMERA, as shown in Table 10. We found that the summary misses the main message, "Views location is not getting updated when switching profiles". The summary just describes the phenomenon such that, "View locations reset I think this happened because I was testing profiles". Similar to other summaries, the summary includes no reproduction steps. Meanwhile, the summary also misses some punctuation symbols. However, compared to PEGASUS, the summarized phrases are more comprehensible.

Table 10. An exampled summary generated by PRIMERA.

View locations reset I think this happened because I was testing profiles.My view locations have been reset. I have Gitlens installed, and I had a few of the gitlens views in their own view container. Now, they are back in their default view container (scm). fixmeMay I know if you are able to reproduce? . . .

In our qualitative comparison of the six models, we found the following. First, only two models, MDNS and BART, included the proper sentence for the bug description, "Views location is not getting updated when switching profiles". Second, two other models, KCMS and PEGASUS, focused on describing reproduction steps without describing a bug. Therefore, users may have a difficulty in capturing the main bug descriptions from the summaries. PRIMERA also started its summary without explaining what the bug is. However, as PRIMERA focused on describing the situation broadly, we can see that the summary of PRIMERA is easier to understand than that of PEGASUS. Last, T5 did not produce desirable results in that it repeated the same word 22 times.

6. Discussion

Section 6.1 discusses the implications of our study, and Section 6.2 discusses the threats to its validity.

6.1. Implication

We conducted an experimental study to apply six summarization models to summarize duplicate bug reports. We tried to combine multiple natural language processing and machine learning techniques, and applied two multi-document summarization methods, KCMS and MDNS. These two methods showed encouraging results. We also modified the existing summarization methods, T5 and BART, to divide an input document into several sections for an effective summarization. We showed that the proposed methods yield better summarization results of bug reports than the state-of-the-art approaches, PEGASUS and PRIMERA. By doing so, we wanted to focus more on our original goal of getting a more informative summary using duplicates.

The implication of our study for researchers, developers, companies, or users is as follows. First, researchers could refer to our experimental results to know which techniques performed well for summarizing duplicate bug reports. Researchers could develop more accurate summarization techniques for duplicate bug reports based on the information provided by this paper. Furthermore, as we did in our experiment, researchers could divide an input document into several sections or classify the sentences in the input document for a better summarization result. That is, researchers could consider input partitioning and classification methods that are specific to summarizing bug reports. In addition, they could consider ensemble methods or the combinations of existing methods to improve summary quality, as we did with KCMS and MDNS.

For developers, we expect to save developers' time in reviewing multiple bug reports by continuing to develop these summarization techniques for multiple bug reports. Such a summarization result may give an overview of bugs that are relevant to a specific software system. Beyond simply summarizing a single bug report, summarizing multiple bug reports could drive many applications in a helpful way to developers. In this perspective, the effectiveness of summary results in helping developers should be validated in future work. Furthermore, it should be explored what kind of summary report will fulfill this expectation needs so that developers will benefit greatly from bug summary results.

Finally, companies or users can utilize these summarization techniques to identify bug trends to support product, operations, and engineering teams. They could standardize bug report forms and categorize bug reports for future actions. To help them, future researchers should study which parts of a report should be summarized and which should not.

6.2. Threats to Validity

We found two threats to internal validity. The first threat lies in splitting an input document into sections. It is difficult to standardize the split sections. Therefore, there is a risk that the summarization model cannot capture the entire context. The incomplete capture of the entire context could lead to inaccurate summarized outcomes. The second threat is relevant to a golden summary. Human annotators create the golden summary. Human annotators could interpret each bug report differently, depending on their understanding, when creating a golden summary. Two annotators manually create different summaries to mitigate the threat, then cross-check each. We also adopted an unsupervised or pre-trained model to reduce the threat because such a model does not require the usage of a golden summary during a training period.

There are also two threats to external validity. The first threat arises due to the dependency of generating and evaluating summary results on a corpus of bug reports and a golden summary. Because of the dependency limitations, our observed results are hardly extended to a different corpus of bug reports and a different golden summary. The second threat is also relevant to a data source. We selected bug reports from the VS Code project in

GitHub and created 16 duplicated bug reports. Therefore, we need to investigate more sets of bug reports from different projects for generalization.

7. Conclusions

In our study, we identified six summarization models and compared their performance for summarizing duplicate bug reports. In our evaluation, we collected 16 pairs of duplicate bug reports and created the relevant ground truths. Through our experiments, we reported the performance of six summarization models quantitatively and qualitatively. Our experimental results revealed that the extractive multi-document approach based on TF-IDF, MDNS, is the most effective in summarizing duplicate reports. In our quantitative analysis, MDNS showed the highest performance (60% ROUGE-1 and 43% ROUGE-2). In our qualitative analysis, MDNS informatively described the bug with problems, reproduction steps, and solutions.

As future work, we will develop more accurate summarization techniques for duplicate bug reports. In the experiment, we also consider showing the effectiveness of duplicate bug report summarization, compared to single bug report summarization. To be more practical, we will conduct a user study to understand what kind of bug summary meets developers' expectation and saves developers' time in reviewing bug reports.

Author Contributions: Conceptualization, S.M. and S.L.; methodology, S.M. and C.C.P.; software, S.M. and C.C.P.; validation, S.M., C.C.P. and S.L.; formal analysis, C.C.P. and S.L.; investigation, S.L.; resources, C.C.P. and S.M.; data curation, S.M.; writing—original draft preparation, C.C.P.; writing—review and editing, S.M., S.L. and P.J.; visualization, C.C.P.; supervision, S.L.; project administration, S.L.; funding acquisition, S.L. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by the Basic Science Research Program through a National Research Foundation of Korea (NRF) grant funded by the Ministry of Education (NRF-2021R1A2C1094167, RS-2023-00209720). This research was also funded by the "Leaders in Industry-university Cooperation 3.0" Project, supported by the Ministry of Education and National Research Foundation of Korea (LINC3.0-2022-11, 1345356213).

Data Availability Statement: Our duplicate bug report datasets are publicly available at https://github.com/primakashi/duplicate_bug_report.

Conflicts of Interest: The authors declare no conflicts of interest. The funders had no role in the design of the study; in the collection, analyses, or interpretation of data; in the writing of the manuscript; or in the decision to publish the results.

Abbreviations

The following abbreviations are used in this manuscript:

ADS	Authorship Dataset
BART	Bidirectional and Auto-Regressive Transformers
BoW	Bag of Words
BRC	Bug Report Corpus
KCMS	K-means, Centroid-based, MMR, and Sentence Position
MDNS	Multi-Doc News Summarizer
MMR	Maximal Marginal Relevance
ROUGE	Recall-Oriented Understudy for Gisting Evaluation
SDS	Summary Dataset
T5	Text-To-Text Transfer Transformer

Appendix A. The Entire Summary Generated by Each Model

Table A1. Examples of summaries generated by KCMS and MDNS

Model	Summary
KCMS	Enable profiles in settings Restart In one of the windows run “Settings Profiles: Create an Empty Settings Profile” and create a new empty profile See that your new profile gets used in just that window In that same window run “Settings Profiles: Switch” and choose your default profile See that your default profile is applied, but view locations are reset /duplicate Thanks for the steps. root cause is the order of storage events. Reloading does not fix my issue. Found an easy repro: Have 2 windows open (not sure if this step is needed) Disable profiles in settings (not sure if this step is needed) Restart. This happens during storage change with views cache storage change comes first followed by view containers cache storage change.
MDNS	Views location is not getting updated when switching profiles Enable Settings Profiles Create an empty profile P1 and move Timeline and Outline views to secondary sidebar as separate view containers Create an empty profile P2 Switch from P2 to P1 Timeline and Outline views are not moved to secondary sidebar and remain in explorer Refreshing the window fixes it. View locations reset I think this happened because I was testing profiles. Similar to #153655 interesting, looks like this is specific to generated view containers, as moving an entire container from activity bar to secondary side bar is reflected correctly yes. My view locations have been reset. This happens during storage change with views cache storage change comes first followed by view containers cache storage change. Found an easy repro: Have 2 windows open (not sure if this step is needed) Disable profiles in settings (not sure if this step is needed) Restart. Enable profiles in settings Restart In one of the windows run “Settings Profiles: Create an Empty Settings Profile” and create a new empty profile See that your new profile gets used in just that window In that same window run “Settings Profiles: Switch” and choose your default profile See that your default profile is applied, but view locations are reset /duplicate Thanks for the steps. I think you might be hitting this—#154090 @sandy081 I took a look at the steps in the duplicate issue, and it says that reloading fixes the issue. I am planning to look at #154090 next milestone and during then I will cover this too.

Table A2. Examples of summaries generated by T5, BART, PEGASUS, and PRIMERA.

Model	Summary
T5	I think this happened because I was testing profiles. Restart./nI think this happened because I was testing profiles. Restart./nI think this happened because I was testing profiles. Reloading does not fix my issue. I think this happened because I was testing profiles. Reloading does not fix my issue. Reloading does not fix my issue. Reloading does not fix my issue. Reloading does not fix my issue./nI think this happened because I was testing profiles. Reloading does not fix my issue. I think this happened because I was testing profiles. Reloading does not fix my issue. Reloading does not fix my issue. Reloading does not fix my issue. Reloading does not fix my issue./nI am planning to look at #154090 next milestone and during then I will cover this too. root cause is the order of storage events. This happens during storage change with views cache storage change comes first followed by view containers cache storage change . . . . My view locations have been reset. I think this happened because I was testing profiles. My view locations have been reset. in. in.. Reload. Re.. This happens This/n
BART	View locations reset I think this happened because I was testing profiles. I have Gitlens installed, and/nEnable profiles in settings. Restart. Create a new empty profile./nI think you might be hitting this—#154090. I took a look at the steps in the/nViews location is not getting updated when switching profiles. Timeline and Outline views are not moved to secondary sidebar/nyes. root cause is the order of storage events. This happens during storage change with views cache storage change comes/n
PEGASUS	Enable profiles in settings Restart In one of the windows run “Settings Profiles: Create an Empty Settings Profile” and create a new empty profile See that your new profile gets used in just that window In that same window run “Settings Profiles: Switch” and choose your default profile See that your default profile is applied, but view locations are reset/duplicate Thanks for the steps. Views location is not getting updated when profiles switching Enable Settings Profiles Create an empty profile P1 and move Timeline and Outline views to secondary sidebar as separate view containers Create an empty profile P2 Switch from P2 to P1 Timeline and Outline views are not moved to secondary sidebar and remain in explorer Refreshing the window fixes it./n
PRIMERA	View locations reset I think this happened because I was testing profiles.My view locations have been reset. I have Gitlens installed, and I had a few of the gitlens views in their own view container. Now, they are back in their default view container (scm). fixmeMay I know if you are able to reproduce? If so can you please provide steps? Zooming in and out of the view containers is not possible when switching profiles./n

References

1. Jindal, S.G.; Kaur, A. Automatic keyword and sentence-based text summarization for software bug reports. *IEEE Access* **2020**, *8*, 65352–65370. [CrossRef]
2. Bettenburg, N.; Premraj, R.; Zimmermann, T.; Kim, S. Duplicate bug reports considered harmful... really? In Proceedings of the 2008 IEEE International Conference on Software Maintenance, Beijing, China, 28 September–4 October 2008; pp. 337–345.
3. Hao, R.; Feng, Y.; Jones, J.A.; Li, Y.; Chen, Z. CTRAS: Crowdsourced test report aggregation and summarization. In Proceedings of the 2019 IEEE/ACM 41st International Conference on Software Engineering (ICSE), Montreal, QC, Canada, 25–31 May 2019; pp. 900–911.
4. Manh, H.C.; Le Thanh, H.; Minh, T.L. Extractive Multi-Document Summarization Using K-Means, Centroid-Based Method, MMR, and Sentence Position. In Proceedings of the 10th International Symposium on Information and Communication Technology, SoICT ’19, Hanoi, Vietnam, 4–6 December 2019; pp. 29–35. [CrossRef]
5. Reichold, L. Multi-Document News Article Summarizer. Available online: https://github.com/lukereichold/News-Summarizer (accessed on 3 July 2023).
6. Raffel, C.; Shazeer, N.; Roberts, A.; Lee, K.; Narang, S.; Matena, M.; Zhou, Y.; Li, W.; Liu, P.J. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* **2020**, *21*, 5485–5551.
7. Lewis, M.; Liu, Y.; Goyal, N.; Ghazvininejad, M.; Mohamed, A.; Levy, O.; Stoyanov, V.; Zettlemoyer, L. BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. *arXiv* **2019**. [CrossRef]
8. Zhang, J.; Zhao, Y.; Saleh, M.; Liu, P.J. PEGASUS: Pre-training with Extracted Gap-sentences for Abstractive Summarization. *arXiv* **2020**. [CrossRef]
9. Xiao, W.; Beltagy, I.; Carenini, G.; Cohan, A. PRIMERA: Pyramid-based Masked Sentence Pre-training for Multi-document Summarization. *arXiv* **2022**. [CrossRef]

10. Yadav, D.; Lalit, N.; Kaushik, R.; Singh, Y.; Dinesh, M.; Yadav, A.K.; Bhadane, K.; Kumar, A.; Khan, B. Qualitative Analysis of Text Summarization Techniques and Its Applications in Health Domain. *Comput. Intell. Neurosci.* **2022**, *2022*, 3411881. [CrossRef] [PubMed]
11. Kumarasamy Mani, S.K.; Catherine, R.; Sinha, V.; Dubey, A. AUSUM: Approach for unsupervised bug report summarization. In Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software, Cary, NC, USA, 11–16 November 2012; p. 11. [CrossRef]
12. Rastkar, S.; Murphy, G.C.; Murray, G. Automatic Summarization of Bug Reports. *IEEE Trans. Softw. Eng.* **2014**, *40*, 366–380. [CrossRef]
13. Lotufo, R.; Malik, Z.; Czarnecki, K. Modelling the 'Hurried' bug report reading process to summarize bug reports. In Proceedings of the 2012 28th IEEE International Conference on Software Maintenance (ICSM), Trento, Italy, 23–28 September 2012; Volume 20, pp. 430–439. [CrossRef]
14. Ferreira, I.; Cirilo, E.; Vieira, V.; Mourão, F. Bug Report Summarization: An Evaluation of Ranking Techniques. In Proceedings of the 2016 X Brazilian Symposium on Software Components, Architectures and Reuse (SBCARS), Maringz, Brazil, 19–20 September 2016; pp. 101–110. [CrossRef]
15. Yang, C.Z.; Ao, C.M.; Chung, Y.H. Towards an Improvement of Bug Report Summarization Using Two-Layer Semantic Information. *IEICE Trans. Inf. Syst.* **2018**, *E101.D*, 1743–1750. [CrossRef]
16. Liu, H.; Yu, Y.; Li, S.; Guo, Y.; Wang, D.; Mao, X. BugSum: Deep Context Understanding for Bug Report Summarization. In Proceedings of the 28th International Conference on Program Comprehension, ICPC'20, Seoul, Republic of Korea, 13–15 July 2020; pp. 94–105. [CrossRef]
17. Kukkar, A.; Mohana, R. Bug Report Summarization by Using Swarm Intelligence Approaches. *Recent Patents Comput. Sci.* **1969**, *12*, 1–15. [CrossRef]
18. Shastri, A.; Saini, N.; Saha, S.; Mishra, S. MEABRS: A Multi-objective Evolutionary Framework for Software Bug Report Summarization. In Proceedings of the 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Melbourne, Australia, 17–20 October 2021; pp. 2006–2011. [CrossRef]
19. Koh, Y.; Kang, S.; Lee, S. Deep Learning-Based Bug Report Summarization Using Sentence Significance Factors. *Appl. Sci.* **2022**, *12*, 5854. [CrossRef]
20. Huai, B.; Li, W.; Wu, Q.; Wang, M. Mining Intentions to Improve Bug Report Summarization. In Proceedings of the International Conferences on Software Engineering and Knowledge Engineering, Redwood City, CA, USA, 1–3 July 2018; pp. 320–363. [CrossRef]
21. Tian, X.; Wu, J.; Yang, G. BUG-T5: A Transformer-based Automatic Title Generation Method for Bug Reports. In Proceedings of the 2022 3rd International Conference on Big Data & Artificial Intelligence & Software Engineering, Guangzhou, China, 21–23 October 2022; Volume 3304, pp. 45–50.
22. Ma, X.; Keung, J.W.; Yu, X.; Zou, H.; Zhang, J.; Li, Y. AttSum: A Deep Attention-Based Summarization Model for Bug Report Title Generation. *IEEE Trans. Reliab.* **2023**, 1–15. [CrossRef]
23. He, J.; Nazar, N.; Zhang, J.; Zhang, T.; Ren, Z. PRST: A PageRank-Based Summarization Technique for Summarizing Bug Reports with Duplicates. *Int. J. Softw. Eng. Knowl. Eng.* **2017**, *27*, 869–896. [CrossRef]
24. Kim, B.; Kang, S.; Lee, S. A Weighted PageRank-Based Bug Report Summarization Method Using Bug Report Relationships. *Appl. Sci.* **2019**, *9*, 5427. [CrossRef]
25. Wang, A.; Singh, A.; Michael, J.; Hill, F.; Levy, O.; Bowman, S.R. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. *arXiv* **2018**. [CrossRef]