A Combination of Classification and Summarization Techniques for Bug Report Summarization

Samal Mukhtar[1,*], Seonah Lee[2,*]

[1]Department of AI Convergence Engineering, Gyeongsang National University, 501 Jinjudaero, Jinju-si, Gyeongsangnam-do, CO 52828 Republic of Korea

[2]Department of Aerospace and Software Engineering and Department of AI Convergence Engineering, Gyeongsang National University, 501 Jinjudaero, Jinju-si, Gyeongsangnam-do, CO 52828 Republic of Korea

Abstract

Well-written bug reports should encompass bug descriptions, reproduction steps, environment details, and solutions. Bug report summaries also need to include such information to be highly informative for developers. However, traditional bug summarization techniques only apply summarization techniques to bug reports, and the generated summaries do not contain such information in a balanced way. In this paper, we propose summarizing duplicate bug reports by including bug descriptions, reproduction steps, environment details, and solutions. For that, our approach combines a supervised classification approach with the pre-trained summarization model BART. Additionally, we performed comparative experiments to demonstrate the effectiveness of this new approach in comparison to existing Summary and Authorship datasets. The experiments reveal that our approach outperforms the state-of-the-art method, achieving a 5% and 7% higher F-score for the Summary and Authorship datasets.

Keywords

bug report summarization, text classification, pre-trained text summarization, classification

1. Introduction

Effective bug reporting is crucial in software development, requiring comprehensive bug reports that encompass detailed bug descriptions, reproduction steps, environmental specifics, and proposed solutions. Bug report summaries, often the first point of reference for developers, should similarly provide this essential information.

Previous work considered observed bug behaviors, steps to bug reproduction, and possible solutions as the key information[1, 2]. However, these approaches often lack a focus on generating summaries with precisely defined attributes that contribute to the informativeness of the bug report. Instead, they tend to prioritize creating general and straightforward summaries.

This paper presents an innovative approach to summarizing bug reports and generating structured summaries, ensuring that specific attributes are extracted and presented comprehensively. In that case, well-organized bug report summaries containing essential information may help to navigate lengthy bug reports quickly[3].

ISE 2023: 2nd International Workshop on Intelligent Software Engineering, Dec 04, 2023, Seoul

*Corresponding author.

aurelisea@gmail.com (S. Mukhtar); saleese@gnu.ac.kr (S. Lee)

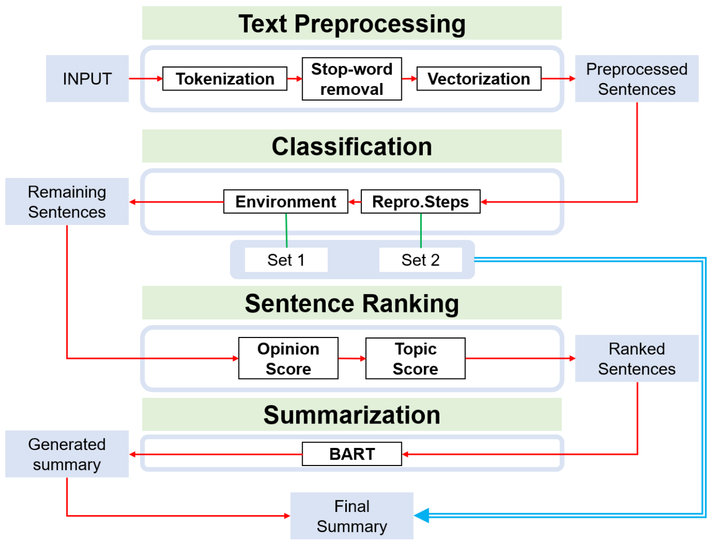

Figure 1: Scheme of the proposed approach

In our work, we describe four attributes to be included in final bug report summaries: Description and the possible Solutions of the bug, its Environment, and Reproduction steps. Thus, we provide a new approach to generate summaries where specific details are extracted and presented in a structured way. The approach would be convenient for developers to process clear, organized information.

Initially, we employ a supervised classification method with domain-specific features to extract environment and reproduction steps from the original bug report. Subsequently, we calculate scores for each sentence in the remaining text after classification. We use these scores to rank and select candidates for inclusion in the final summary, generated using the pre-trained summarization model BART[4].

To assess our approach's effectiveness, we evaluated accuracy and readability across two datasets, including the widely-used bug report Summary (SDS) and Authorship (ADS) datasets[5, 6]. Our results demonstrate that our approach achieved remarkable improvements, with 55% and 56% in terms of F-score for SDS and ADS, respectively, showing a higher accuracy than the baseline model.

The rest of this paper is organized as follows. Section 2 describes the Related Works. Sections 3 and 4 describe the Proposed Approach and the Experimental Setup. Section 5 discusses the results. We conclude in Section 6.

2. Related Work

Rastkar et al.[2] initiated bug report summarization research in 2010, targeting the optimization of project artifacts for developers. They focused on issue reports, believing that better summarization could help in identifying duplicate bug reports. Their unique perspective treated bug reports as dialogues, using machine-learning classifiers (EC, EMC, and BRC) trained on different conversational data sources for effective summarization. BRC proved the most effective in evaluation metrics and human coherence. The study's contributions include a bug corpus creation and a novel approach for demonstrating regression model effectiveness in bug report summarization.

Rastkar et al. consider classifiers as an approach and uses them directly for summarization, while in our work, we utilize classifiers as a pre-step for summarization. Notably, we evaluate our data on a bug corpus provided by their study.

Mani et al.[7] challenge the efficiency of supervised models for summary generation, emphasizing their reliance on large data and training limitations. They introduce an innovative unsupervised approach comprising noise removal and summarization stages. Before summarizing issue reports, the original text is categorized into groups like Question, Code, Investigation, and Others. Their classifier module employs four methods: Centroid, Maximum Marginal Relevance (MMR), Grasshopper, and Different Rank. Their evaluation demonstrates that unsupervised techniques outperform the supervised method[2] in terms of total F-score. This work offers two significant contributions: showcasing the effectiveness of noise removal before summarization and the first instance of unsupervised methods surpassing supervised ones in summarizing issue reports. However, precision remains a limitation, suggesting potential inaccuracies in sentence classification and a strong reliance on the filtering mechanism.

In this research, we also depend on a noise removal stage. However, the classification methods we employ differ in terms of techniques and methodology. While we classify sentences in bug reports based on their attribute consideration, Mani et al. focus on the inherent features of the sentences themselves.

In the context of mining software repositories (MSRs), the effectiveness of controlled approaches strongly relies on attributes, which differentiate data. However, no systematic analysis has examined how to create new attributes for bug report summarization. Jiang et al. [8] introduce a unique method involving crowdsourcing for supervised summarization, identifying 11 new text features. Unlike prior work[2], the authors emphasize the crowdsourcing process in selecting these features and explain their choices. This approach and the text features they introduce prove to be more effective. To train a statistical model, specifically logistic regression, they evaluate these 11 attributes on the training data. They use a trained model that considers sentence attributes to make a summary. While the authors suggest focusing on specific features for better efficiency, relying heavily on crowdsourcing can complicate the application to summarization, as most works in this field typically need more crowdsourced attributes.

In their study, they employ various ranking techniques to identify textual similarities among multiple sources using feature-based approaches. In contrast, we prioritize ranking sentences based on their topic relevance, thus selecting candidates for subsequent summarization.

Li et al. [9] underscore the significance of bug report characteristics, which they believe can affect summarization technique effectiveness. Bug reports involve conversations and include different types of sentences in both natural and software languages. They represent the predefined fields with important information for identifying key sentences. To solve these challenges, they introduce the DeepSum approach, an unsupervised auto-encoder model. DeepSum leverages deep neural network hidden layers to compress input feature vectors from bug report sentences and selects summary sentences based on the weights assigned by sentence selection algorithms. Compared with the current study, these are based on sentence types of bug reports, whether they are written in natural language or software sentence language. Extensive experiments demonstrate that DeepSum significantly outperforms comparative algorithms [2, 7] in F-score and Rouge metrics. This approach marks the first use of deep learning in bug report summarization. Nevertheless, it has limitations because text features are not controlled, and sentence weights depend on vector frequency, which may only sometimes guarantee high accuracy.

Huai et al.[10] introduced an intention-based bug report summarization approach (IBRS), defining seven key intention categories within bug reports. They expanded the Ratskar BRC corpora into Intention-BRC for their evaluation, examining the connection between bug report summaries and the underlying sentence intentions. This strategy led to an improved bug report summarization method incorporating sentence intentions. The authors established an intention taxonomy for bug reports, categorizing intentions into seven groups: Bug Description, Fix Solution, Opinion Expressed, Information Seeking, Information Giving, Meta/Code, and Emotion Expressed. They proposed a mixed model combining linguistic pattern matching and machine learning to classify sentences based on their intentions. IBRS leverages implicit information within sentence intentions, potentially enhancing bug report summarization in software development and debugging. However, the authors must consider factors such as extensive data and validation, potential computational complexity, and the risk of readability and information retention issues in summary rearrangement.

In contrast to Huai et al.'s intention-based approach, our work adopts a classifier-based strategy, prioritizing the extraction and summarization of bug report attributes rather than sentence intentions. Additionally, while Huai et al. emphasize the connection between bug report summaries and sentence intentions, our research centers on the relevance of sentences to the issue report topic, resulting in distinct approaches to bug report summarization.

Nawaz Tarar et al.[11] introduce a technology akin to the DeepSum approach[9], employing an auto-encoder. Their method calculates sentence similarity through sentence embedding, followed by K-means clustering to organize sentences into clusters. Sentence ranking is utilized to select high-ranking sentences based on their information content for summarization. The critical difference between their work and ours is the utilization of different baselines in sentence selection. They are incorporating additional clustering techniques resulting in significantly better F-score and Rouge metric performance compared to previous approaches [2, 7, 9].

Liu et al.[1] address issues stemming from word frequency-reliant methods, which can introduce bias. They introduce BugSum, an unsupervised approach leveraging deep learning networks. BugSum combines an auto-encoder network for feature extraction with a novel believability metric to gauge sentence approval within discussions. Initially, they extract semantic features using the auto-encoder network and calculate the degree of approval during discussions. The approach combines a believability score with a vector for each sentence to prevent the inclusion of controversial sentences in the summary. This method surpasses the previous approach[8], but its reliance on evaluated sentences limits its accuracy since not all bug reports contain such evaluations.

Although, in the current study, we utilize the same score presented in the work of Liu et al., we use it in different contexts of reducing the number of noisy sentences with a combination of topic relevance scores. Thus, we can significantly reduce the number of unpleasant sentences.

Gupta et al.[12] emphasize the significance of timestamps, sentences related to the topic, authorship, and noise filtering. They dissect textual features to separately select comments and sentences based on hypotheses, ultimately categorizing the extracted features into comment-specific and sentence-specific features. They prioritize finding critical comments first and then apply selective sentence-specific features to identify crucial sentences within those comments. Thus, the authors paid attention to selecting sentences regarding some sentence specifications. Combining description and comment summaries enriches the overall summaries. In contrast, we emphasize the relevance of sentences to the issue report topic and use weight assignment techniques to filter noise. However, the approach's dependency on the dictionary can be a weakness, as bug reports often contain references, jargon, and domain-specific language that may need to be adequately covered by the chosen dictionary, potentially leading to subpar summaries.

Koh et al.[13] address the limitations of existing bug report summarization methods, which rely on training data or word frequencies and often need to improve in producing high-quality summaries. They aim to provide a more versatile and practical approach by introducing sentence significance factors. These factors serve as criteria for identifying meaningful sentences in bug reports. The proposed deep-learning-based method utilizes sentence-to-sentence cohesion, topic association, and believability. In contrast, our approach combines the topic association factor with additional score assessment to identify sentences that align with bug description and behavior. Experimental results demonstrate that their approach outperforms the state-of-the-art BugSum[1] in terms of precision, recall, and F-score. This work makes two key contributions: it presents a comprehensive deep-learning-based bug report summarization method that incorporates various sentence significance factors and thoroughly compares with BugSum using two benchmark datasets, highlighting the superiority of the approach employing three sentence significance factors over BugSum.

In their work, Demeter et al.[14] presented an insightful analysis of customer-facing summarization systems, emphasizing the superior performance of BART over PEGASUS and T5 in both automatic and human evaluations. They discovered that summarizers exhibit significantly diminished performance when applied across different domains. However, a noteworthy finding was that a system fine-tuned on heterogeneous domains displayed robust performance across all domains, rendering it an ideal choice for a broad-domain summarizer. This work underscores the necessity for heterogeneous domain summarization benchmarks and unveils substantial variability in system outputs, a phenomenon best discerned through human evaluation, which often eludes standard leaderboards reliant solely on automatic metrics. Furthermore, their human evaluation revealed a strong preference for BART summaries, particularly those trained on mixed data, even surpassing summaries generated by BART fine-tuned on in-domain data, a preference that defies automatic metric capture. These insights solidify BART's position as a leading model for customer-facing summarization systems.

Mercan et al.[15] conducted a study on text summarization, which involves condensing large amounts of text into concise summaries. The research involved training an LSTM model and fine-tuning BART, T5, and PEGASUS using a resume dataset, with BART-Large emerging as the top performer based on the highest ROUGE score. Additionally, they used open-source datasets for training and fine-tuning. These findings highlight BART and BART-Large's significance in text summarization, particularly in resume classification tasks.

Compared to our research, studies of Demeter et al. and Mercan et al. advocate fine-tuning as the primary procedure for harnessing summarization models. Our study emphasizes pre-processing steps preceding summarization to make bug reports more suitable for a generic model, as fine-tuning could require significant resource allocation.

Overall, the novelty of our work lies in the approach to bug report summarization. While previous research has focused on sentence intentions or linguistic patterns, our contribution is summarizing bug reports by explicitly considering their attributes. Although we employ similar scoring and weight assignment techniques that have been utilized in the past, the key innovation is in using a classification process as the primary domain-specific stage. This classification process helps select and filter sentences, ensuring the resulting summaries are highly relevant. Furthermore, by incorporating pre-trained models for the summarization task, we enhance the efficiency and accuracy of the bug report summarization process, contributing to a more effective and targeted approach to software development and debugging.

3. Proposed Approach

Our proposed summarization approach consists of four key steps. To commence, we initiate data preprocessing through three primary stages: removing stop words, word stemming, and vectorization. Following this, we employ a pre-trained random forest classifier to identify sentences within bug reports relevant to the Environment and Reproduction Steps domains.

Subsequently, we proceed with the scoring process for the remaining sentences in the text, implementing an initial noise reduction step. This scoring process relies on two core features, as previously outlined by Koh et al. [13] and Liu et al.[1]. We have observed that combining these two features helps to filter out less crucial sentences, thus enhancing the content used in the summary generation step. In this phase, we utilize a pre-trained BART model. Following our initial classification and sentence ranking stages, we significantly increase the likelihood of generating high-quality summaries through a general text summarization model.

In the final step, we merge the generated summaries with the sentences extracted during the classification process. Figure 1 illustrates the approach scheme.

3.1. Text Preprocessing

Our data preparation process initiates by extracting textual information from bug reports. After this, we perform essential preprocessing steps. Tokenization breaks the text into discrete units, facilitating subsequent analysis. Next, we remove common stop words to streamline the data.

After refining the preprocessed text, we move on to a crucial domain feature extraction phase. Here, we employ BERT-as-a-Service, transforming our text into numerical vectors. These vectors capture essential attributes and intricate patterns, driving our subsequent analysis.

Our meticulous data preparation encompasses extraction, tokenization, stop-word removal, and domain feature extraction. Each step aims to ensure data integrity and utility for our research and analysis.

3.2. Classification

In the second stage of our approach, we leverage a pre-trained Random Forest classifier, as detailed in Section 3 of our methodology. We subject each sentence within the bug reports to meticulous analysis during the classification process, using the preprocessed sentences from the preceding text preprocessing step as inputs.

This step categorizes the sentences into two sets, each associated with the Environment or Reproduction Steps groups. These classified sentences play a pivotal role in our subsequent summary generation process, ensuring we effectively convey essential information related to Environment and Reproduction Steps in the final summaries. Thus, as the output of the second step of the approach, we get three sets of sentences. The first two are sets of classified sentences for Environment and Reproduction Steps, respectively, and the third set is the remaining text sentences. The last one will pass to the following sentence ranking Step, while the first two remain until summaries are generated.

3.2.1. Classification Model

The Random Forest (RF) classifier, a widely adopted machine learning algorithm, functions by constructing an ensemble of decision trees during training[16]. Each decision tree is crafted using a random subset of the training data and a random subset of the features, thereby introducing variability into the individual trees. The ensemble strategy consolidates the predictions from these diverse decision trees, typically through a majority voting mechanism in classification tasks. We employed the RF classifier as our predictive model in our specific classification task for several compelling reasons. First and foremost, its simplicity and ease of integration make it an ideal choice for the task at hand. Random Forest operates effectively without requiring extensive hyperparameter tuning, making it straightforward to implement in the classification framework[17].

Moreover, its versatility and robustness make it a commonly employed algorithm in classification tasks across various domains. The model's ability to handle both categorical and numerical features and its capability to mitigate overfitting further underscores its suitability for binary classification in this context. The primary aim of this model was to acquire insights and identify patterns within a labeled dataset comprising sentences along with corresponding 'Environment' or 'Reproduction Steps' labels.

3.2.2. Our Use of the Classification Model

To enhance the learning process, we introduced a feature extraction method that generated sets of eight and five informative features for each sentence's textual content, corresponding to 'Environment' and 'Reproduction Steps,' respectively. The features were derived from analyzing various bug reports, considering common patterns and tendencies observed in the language used. We chose these features to capture critical attributes such as sentence length, numerical content, and other pertinent characteristics, as detailed in Table 1. The rationale for deriving each feature is grounded in the observation that bug reports often exhibit consistent patterns, such as short lengths in environmental sentences, the inclusion of numerical data, the presence of fixed common phrases and keywords, the absence of questions, and other identifiable traits. While

Table 1
Extracted features

Feature number	Label	Feature Description
1	Environment	To count of words in the sentence.
2	Environment	Presence of a ':' colon in the sentence.
3	Environment	To capture whether the sentence starts with key phrases like "OS version" or "OS" common in bug reports.
4	Environment	To identify documents that match specific keyword criteria while considering document length. (often shorter for environment descriptions).
5	Environment	To identify of key phrases in the sentence, considering the presence of numbers and digits (common in bug reports).
6	Environment	To detect special characters ('|')
7	Environment	To determine whether a given text contains a version number or a product name.
8	Environment, Reproduction Steps	Filtering out sentences that contain question marks, indicating queries or uncertainties.
9	Reproduction Steps	To capture whether a given sentence in a text document has a verb that is directly followed by a direct object.
10	Reproduction Steps	To identify whether a sentence in a text document is written in the imperative mood by checking if it contains a verb in the base form that serves as the root of the sentence's syntactic structure.
11	Reproduction Steps	To identify whether a sentence in a text document contains a verb that matches a specific verb form pattern and is the root of a clause.
12	Reproduction Steps	To identify whether a sentence starts with a pattern that resembles a numerical enumeration or list item.

we acknowledge that individual projects may exhibit unique features, we focused on specifying the most common and evident ones based on our analysis. Subsequently, we partitioned the dataset into training and testing subsets, with 80% of the data allocated for model training and the remaining 20% reserved for model evaluation.

The Random Forest classifier underwent training on the designated training data, enabling it to establish connections between the extracted features and the associated 'Environment' and 'Reproduction Steps' labels. This training process empowered the model to make predictions regarding labels for new sentences based on their feature representations. After training, we serialized the model to a file for future use, and we also saved the feature extraction method to ensure consistent data preprocessing for inference tasks.

3.3. Sentence Ranking

In the subsequent step of our process, we delve into the crucial ranking phase, where we employ two key features sourced from the works of authors [13] and [1]. Our experiments showed how extracting opinion scores proves invaluable in sifting out non-relevant sentences from the bug reports. In contrast, the topic score effectively identifies the most significant sentences concerning the subject of the original bug report. To harmonize these two features, we calculate the average value of their respective scores. However, we also underscore the significance of sentences at both the input's beginning and end. The bug report's description is typically found at the outset, while the text's conclusion contains the solution. Therefore, our comprehensive ranking process involves evaluating and ordering sentences, filtering out the top 40-45% of high-score sentences, and retaining five sentences from the beginning and end. This approach ensures that our generated summaries capture the essence of the bug report while maintaining a balanced structure.

3.4. Text Summarization

In the final step of our process, Summarization, we take the ranked sentences from the previous phase and utilize a pre-trained BART model to generate a concise summary. We aim to create summaries encompassing essential information, including issue descriptions and workarounds with possible solutions. Following the generation of the summary, we seamlessly integrate it with the two sets of classified sentences obtained from the Classification step. This grouping ensures that our summaries capture the core elements of the bug report and provide comprehensive insights into issue descriptions and potential solutions.

As a summarization model, we leverage the capabilities of the Bidirectional and Auto-Regressive Transformers model (BART). BART is a pre-trained transformer-based model known for its proficiency in various text generation tasks, particularly in abstractive text summarization. While we can perform fine-tuning processes on BART for tasks specific to particular domains, like bug report summarization, we opt not to engage in this pre-training and fine-tuning procedure. We decided because the first two steps effectively addressed specific aspects of the bug reports, making the input data readily understandable for the model after this initial cleaning process.

4. Experimental Setup

4.1. Research Questions

- **RQ1. What is the performance of our approach in classifying bug report attributes?**
  Our initial research inquiry is centered on assessing the performance of our proposed approach in classification tasks. We aim to evaluate its accuracy and effectiveness in classifying data and compare its performance against established benchmarks. This evaluation is essential for gauging its suitability in real-world classification scenarios.
- **RQ2. How effectively does our approach perform in terms of bug report summarization?**
  We seek to assess the clarity and comprehensibility of the summaries generated by our approach and compare them to existing summarization baselines. This evaluation is essential to gauge the practical usefulness of our approach in conveying information clearly and effectively.

4.2. Dataset

To show the effectiveness of the proposed approach, we conduct an evaluation procedure with two Summary (SDS) and Authorship (ADS) Datasets. Thus, we aim to show comparative results and the reliability of the approach. Datasets comprise 36 and 96 bug reports from four open-source projects, namely Eclipse, Mozilla, KDE, and Gnome, each for SDS and ADS, respectively.

To make the evaluation possible, we modified the ground truths of each Dataset. Earlier works in bug report summarization did not consider attributes of bug reports separately as we did in our current work. Thus, there is no ground truth for them. To mitigate this, we add sentences that may contribute to “Environment” or “Reproduction Steps” attributes in both SDS and ADS datasets, modifying the golden summaries to contain original ground truth with an extended set of sentences.

4.3. Baseline Model

We primarily rely on BugSum [1], an unsupervised approach based on a deep learning network, a well-established baseline that has outperformed other bug report summarization methods in previous comparisons[3] [9]. The bug summarization method integrates the auto-encoder network for feature extraction, employs a unique sentence believability assessment, and implements dynamic selection techniques.

4.4. Experimental Procedure

In response to research question **RQ1**, our primary goal is to demonstrate the effectiveness of our chosen approach in classification. We gather classified sentences alongside the previously identified golden sentences in the same file for evaluation. This approach allows us to compute the accuracy for each bug report, and we subsequently determine the average accuracy across all 21 bug reports. We assess performance using three key metrics: Precision, Recall, and the F-Score.

For research question **RQ2**, our focus is on assessing the accuracy of the generated summaries. After generating the summaries, we combine them with the previously classified sentences from the initial phase, resulting in final summaries that consist of three essential segments. We then utilize the ROUGE toolkit to obtain numerical evaluation results for each bug report and calculate the Accuracy scores. In the final step, we calculate the entire dataset’s average readability and accuracy scores.

4.5. Evaluation Metrics

Recall measures the effectiveness of a generated summary in capturing and encompassing vital details from the reference summary. It quantifies the ratio of pertinent content in the generated summary to the total pertinent content within the reference summary. In the context of bug report summaries, a high recall reflects the capacity to accurately capture and integrate essential information.

$$Recall = \frac{Chosen \bigcap GSS}{Chosen} \tag{1}$$

Precision assesses the accuracy and relevance of information in a summary. It determines the ratio of pertinent content in the summary to the total content. In the context of bug report summaries, high precision signifies that the summary effectively concentrates on crucial details of the bug report.

$$Precision = \frac{Chosen \bigcap GSS}{GSS} \tag{2}$$

The F1 score offers a well-rounded evaluation by considering both precision and recall. It represents the harmonic mean of these two metrics and contributes to an all-encompassing assessment of the quality and effectiveness of summaries, bug report summaries included. This balance is evaluated comprehensively, considering the interplay between precision and recall.

$$F - score = \frac{2 * Precision * Recall}{Precision + Recall} \tag{3}$$

These three metrics above are calculated from the selected sentence set *Chosen* and the golden standard sentence set *GSS* to measure the accuracy of the summaries.

We apply the ROUGE toolkit to evaluate our experimental results from the position of readability. The tool measures the quality of generated summaries by counting continuously overlapping units between the summaries and the ground truth. As n-gram ceiling units in the ground truth, we use 1 and 2.

$$Rouge - n = \frac{\sum_{s \epsilon GT} \sum_{n-gram \epsilon s} Count_{match}(n - gram)}{\sum_{s \epsilon GT - gram \epsilon s} Count(n - gram)} \tag{4}$$

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.

5. Experimental Results

5.1. Experimental Results for RQ1

Table 2 displays the experimental results for classification, showing varying performance across different attributes and datasets. When classifying the Environment attribute in the SDS dataset, the approach achieved a precision of 0.75 and a recall of 0.93, resulting in an F-score of 0.73. Similar precision (0.84) and recall (0.83) values in the ADS dataset led to the same F-score of 0.73. However, for the Reproduction Steps attribute, the SDS dataset yielded a precision of 0.81 but a lower recall of 0.55, resulting in an F-score of 0.5. In contrast, in the ADS dataset, the precision (0.84) was higher, and the recall (0.91) was significantly better, resulting in a notably higher F-score of 0.77. The inherent characteristics of each dataset contribute to the discrepancy.

The SDS dataset comprises 36 bug reports, primarily characterized by a higher noise level. These reports often contain sentences from patches within the bug report, making them prone to misclassification as 'Environment' or 'Reproduction Steps' sentences. As a result, the classification model on the SDS dataset demonstrated a lower recall (0.55), indicating a higher likelihood of misclassifying sentences and counting non-sentences in the 'Reproduction Steps' category.

In contrast, the ADS dataset comprises 96 bug reports that, in comparison to the SDS dataset, are generally shorter, predominantly written in natural language, and frequently lack explicit information about the 'Environment' and 'Reproduction Steps.' This characteristic of the ADS dataset prevents misclassification and contributes to a higher recall (0.91), leading to a significantly better F-score of 0.77. These results underline the influence of dataset quality on classification performance, and the presence of noisy bug reports may contribute to lower recall rates.

Table 2
Classification evaluation results

	SDS		**ADS**	
Dataset	Environment	Reproduction Steps	Environment	Reproduction Steps
Precision	0.75	0.81	0.84	0.84
Recall	0.93	0.55	0.83	0.91
Fscore	0.73	0.50	0.73	0.77

5.2. Experimental Results for RQ2

Table 3 displays the outcomes of our experiments on the SDS and ADS datasets. We compared the performance of our proposed approach with the primary baseline model, BugSum, and included both the original BugSum results and our results from their open-source project. In the table, the Proposed Approach exhibits a significant increase in accuracy, achieving 55% and 56% for SDS and ADS, respectively. It highlights its ability to capture a larger share of relevant information within bug reports.

Table 3
Evaluation results

	SDS			**ADS**		
Dataset	Bug-Sum	Bug-Sum (our)	Proposed Approach	Bug-Sum	Bug-Sum (our)	Proposed Approach
Precision	0.63	0.50	**0.68**	0.61	0.51	**0.63**
Recall	0.41	0.43	**0.49**	0.42	0.51	**0.52**
Fscore	0.49	0.44	**0.55**	0.49	0.48	**0.56**
Rouge-1	**0.59**	0.44	0.50	**0.56**	0.31	0.54
Rouge-2	0.19	0.15	**0.24**	**0.27**	0.11	0.24

However, when considering the Rouge-n results, the Proposed Approach registers relatively lower scores when compared to the original BugSum. The reduced Rouge-1 and Rouge-2 scores imply that the generated summaries may not faithfully replicate the exact phrases and sentences found in the original bug reports. It is essential to highlight that this contrast arises because the Proposed Approach utilizes an abstractive methodology, while BugSum employs an extractive one.

The evaluation results demonstrate that the Proposed Approach excels in bug report summarization, exhibiting superior precision, recall, and F-score. These results underscore the effectiveness of the Proposed Approach in generating informative and relevant summaries and suggest its suitability for the task at hand.

6. Conclusion

Overall, this paper has introduced an innovative approach to bug report summarization designed to extract and present specific attributes comprehensively. Our approach initiates with the application of a supervised classification method, leveraging domain-specific features to extract environment and reproduction steps from the original bug report. Subsequently, sentence scores are computed for the remaining content post-classification, and these scores are employed for ranking and selecting candidates to be included in the final summary, generated with the aid of the pre-trained summarization model, BART.

To validate the effectiveness of our approach, we conducted a comprehensive evaluation across two datasets, specifically the widely-used bug report SDS and ADS datasets. Our results have proven promising, with remarkable improvements demonstrated, showing an impressive increase of 5% and 7% in F-scores for SDS and ADS, respectively. This robust performance underscores the superior accuracy of our approach when compared to baseline models. Overall, this work represents a significant step in enhancing bug report summarization efficiency and precision in the software development landscape.

References

[1] H. Liu, Y. Yu, S. Li, Y. Guo, D. Wang, X. Mao, Bugsum: Deep context understanding for bug report summarization (2020) 94–105. URL: https://doi.org/10.1145/3387904.3389272. doi:10.1145/3387904.3389272.

[2] S. Rastkar, G. Murphy, G. Murray, Summarizing software artifacts: A case study of bug reports 1 (2010) 505–514. doi:10.1145/1806799.1806872.

[3] R. Lotufo, Z. Malik, K. Czarnecki, Modelling the 'hurried' bug report reading process to summarize bug reports, volume 20, 2012, pp. 430–439. doi:10.1109/ICSM.2012.6405303.

[4] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, L. Zettlemoyer, BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension, CoRR abs/1910.13461 (2019). URL: http://arxiv.org/abs/1910.13461. arXiv:1910.13461.

[5] S. Rastkar, G. C. Murphy, G. Murray, Automatic summarization of bug reports, IEEE Transactions on Software Engineering 40 (2014) 366–380. doi:10.1109/TSE.2013.2297712.

[6] J. He, J. Zhang, H. Ma, N. Nazar, Z. Ren, Mining authorship characteristics in bug repositories, Sciece China. Information Sciences 60 (2017) 1–16. doi:10.1007/s11432-014-0372-y.

[7] S. K. Kumarasamy Mani, R. Catherine, V. Sinha, A. Dubey, Ausum: Approach for unsupervised bug report summarization, 2012, p. 11. doi:10.1145/2393596.2393607.

[8] J. He, N. Nazar, J. Zhang, T. Zhang, Z. Ren, Prst: A pagerank-based summarization technique for summarizing bug reports with duplicates, International Journal of Software Engineering and Knowledge Engineering 27 (2017) 869–896. doi:10.1142/S0218194017500322.

[9] X. Li, H. Jiang, D. Liu, Z. Ren, G. Li, Unsupervised deep bug report summarization (2018) 144–14411.

[10] B. Huai, W. Li, Q. Wu, M. Wang, Mining intentions to improve bug report summarization, 2018, pp. 320–363. doi:10.18293/SEKE2018-096.

[11] M. I. Nawaz Tarar, F. Ahmed, W. H. Butt, Automated summarization of bug reports to speed-up software development/maintenance process by using natural language processing (nlp) (2020) 483–488. doi:10.1109/ICCSE49874.2020.9201846.

[12] S. Gupta, S. Gupta, An approach to generate the bug report summaries using two-level feature extraction, Expert Systems with Applications 176 (2021) 114816. doi:10.1016/j.eswa.2021.114816.

[13] Y. Koh, S. Kang, S. Lee, Deep learning-based bug report summarization using sentence significance factors, Applied Sciences 12 (2022). URL: https://www.mdpi.com/2076-3417/12/12/5854. doi:10.3390/app12125854.

[14] D. Demeter, O. Agarwal, S. B. Igeri, M. Sterbentz, N. P. Molino, J. M. Conroy, A. Nenkova, Summarization from leaderboards to practice: Choosing A representation backbone and ensuring robustness, CoRR abs/2306.10555 (2023). URL: https://doi.org/10.48550/arXiv.2306.10555. doi:10.48550/arXiv.2306.10555. arXiv:2306.10555.

[15] �Mercan, S. Cavsak, A. Deliahmetoglu, S. Tanberk, Abstractive text summarization for resumes with cutting edge nlp transformers and lstm (2023).

[16] V. Kulkarni, P. Sinha, Random forest classifiers: A survey and future research directions, International Journal of Advanced Computing 36 (2013) 1144–1153.

[17] N. Kanvinde, A. Gupta, R. Joshi, Binary classification for high dimensional data using supervised non-parametric ensemble method (2022).