Improving Bug Reporting by Fine-Tuning the T5 Model: An Evaluation in a Software Industry

Davi Gonzaga*
Sidia Institute of Science
Manaus, Brazil
davi.menezes@sidia.com

Leonardo Tiago*
Sidia Institute of Science
Manaus, Brazil
leonardo.albuquerque@sidia.com

Ana Paula Silva
Sidia Institute of Science
Manaus, Brazil
silva.ana@sidia.com

Flávia Oliveira
Sidia Institute of Science
Manaus, Brazil
flavia.oliveira@sidia.com

Lennon Chaves
Sidia Institute of Science
Manaus, Brazil
lennon.chaves@sidia.com

ABSTRACT

Context: Bug reporting is essential in software development to ensure product quality. Within testing teams, testers report multiple bugs daily through manual reports describing test details. **Problem:** Manual bug reporting is exhausting and time-consuming, leading to potential errors and missing information. These issues can compromise the product if developers cannot understand the bug's cause. **Solution:** We explore training Large Language Models (LLMs) to automatically generate bug reports from brief user descriptions. **Methodology:** We collected 1,800 bugs reported in 2024 to build a dataset and fine-tuned the T5 model (Text-to-Text Transfer Transformer). Three fine-tuning iterations were performed and evaluated using BLEU, METEOR, Precision, Recall, and F1-Score metrics. **Summary of Results:** The best-performing model, trained with two low-complexity bug report types, achieved BLEU of 0.9845, METEOR of 0.9634, Precision of 0.9898, Recall of 0.9886, and F1-Score of 0.9892. In testing, this model achieved 70% success rate with responses matching user input without hallucinations. The results indicate LLMs are viable for automatic bug report generation.

KEYWORDS

Bug Report, Large Language Models, Software Testing

1 Introduction

The defect reporting stage is a crucial part for software developers to correct all the defects present during development, so that the product does not reach the end user with issues [4]. The process of reporting defects tends to be long and complex, which makes it prone to human errors [11]. However, due to the evolution of Large Language Models (LLMs) [10], it has been possible to optimize several processes in the field of software engineering, including steps from the verification and validation stage.

In a testing team of the software industry, several mobile device projects are tested every day, many bugs are found, and all of them need to be manually reported to the respective developers, which consumes a lot of time and requires great attention to avoid errors. Due to the need to include various types of information (such as reproduction steps, problem details and testing context), it is necessary to pay close attention to ensure that all details are included concisely, so that developers can understand the problem and find a solution efficiently [3].

*Both authors contributed equally in this research.

In order to automate the process of bug reporting, we propose the use of LLMs trained to automatically generate complete bug reports based on prompts that contain a brief description of the problem. To achieve this goal, we first built a dataset with 1800 bug reports created in 2024 by the testing team of the software industry. Through this study, we aim to show how it is possible to automate the bug reporting process using LLMs, as well as to demonstrate, in general terms, that LLMs can contribute to the execution of Software Engineering tasks, in our context, bug reporting.

2 Methodology

The goal of this study is to automate the process of bug reporting using LLMs. To achieve this goal, we followed the methodology below:

(1) **Dataset Preparation**: We built a dataset composed of 1,800 examples of distinct bugs reported in 2024.
(2) **Models and Architectures**: We studied and evaluated different types of architectures for the language model we intended to develop.
(3) **First Fine-Tuning Round**: We performed one fine-tuning iteration using the chosen architecture and the constructed dataset.
(4) **Metrics**: We researched the most suitable metrics for the type of problem we are trying to solve.
(5) **Second Fine-Tuning Round**: We performed another fine-tuning iteration using a more complex model, with updated dataset and metrics, followed by a test carried out by a member of the testing team.
(6) **Third Fine-Tuning Round**: We performed a final fine-tuning iteration using models dedicated to specific groups of bugs and new datasets tailored to each model, followed by another test conducted by a tester.

In the following subsections, we provide details on how each step of this methodology was carried out.

2.1 Dataset Preparation

2.1.1 Building the Dataset. We selected and grouped 1,800 examples of bug reports in order to build a complete dataset, this dataset was then split into two parts, 1,500 bug reports for the training set and 300 for the test set. All the bug reports were reviewed by a senior software tester which ensured the quality and completeness of our dataset.

2.1.2 Processing the Data. The bug reporting standard specifies that reports written by testers must contain 7 fields, which are:

- **Title**: A brief description of the bug, including tags that indicate the scenario in which the bug was found.
- **Problem**: A detailed description of the bug and the scenario in which it occurred.
- **Preconditions**: A list of conditions that must be met before executing the steps to reproduce the bug.
- **Steps to Reproduce**: A list of steps to recreate the scenario where the bug was found.
- **Expected Result**: What was expected to happen instead of the failure.
- **Additional Information**: Details about the test scenario, such as the type of device, the operating system version it was running, the type of test being performed, etc.
- **Binary File Information**: Specifications of the binary files uploaded to the test device for test execution.

Bug reports may contain private details about test devices or tester context. This information requires pre-processing to remove irrelevant content. The "Title" field's identifying tags were standardized across the dataset. The Additional Information field's test environment details were removed. The remaining fields contain only bug report information in a consistent pattern.

2.2 Models and Architectures

The decision regarding the model to be used as the basis for performing the experiments in this study was made based on a review of relevant architectures found in the literature. The architectures we studied are described below:

- **Next Word Prediction**: An approach based on a framework of encoding and decoding, integrated attention mechanisms, and reinforcement learning techniques [9].
- **Questions and Answers**: An approach for obtaining answers to questions by providing long contexts as input to LLMs [1].
- **Seq2Seq**: An approach based on sequential processing architectures, that is, models that take sequences as input and produce sequences as output [8].

After studying and evaluating the feasibility of implementing these architectures, as well as determining which one best suited our work, we chose to proceed with the Seq2Seq architecture. For this reason, the model selected as the foundation for this study was the Text-to-Text Transfer Transformer, or T5, which treats all NLP tasks as text-to-text tasks and is designed to handle various types of sequence-based problems [7].

2.3 First Fine-Tuning Round

2.3.1 Fine-Tuning Strategy. Initially, we had only 6GB of GPU available for this work, so a configuration strategy was adopted to avoid memory overflow during the process. In this first experiment we used only the accuracy metric to evaluate the model's performance as it is commonly used in the training of neural network models.

2.3.2 Fine-Tuning Execution. The first fine-tuning used the "T5-Small" model with 60 million parameters on 1,800 bugs divided into 1,500 training and 300 test examples. We concluded accuracy was not the most appropriate metric for evaluating text generation model performance. Therefore, before conducting further experiments, we researched more suitable metrics.

2.4 Metrics

After the first fine-tuning round, we focused on defining evaluation metrics for the model. We decided to implement 3 metrics considered more suitable for evaluating an LLM, presented below:

- **BLEU**: BLEU (Bilingual Evaluation Understudy) compares model output with human references and calculates similarity scores based on matches [6].
- **ROUGE**: ROUGE (Recall-Oriented Understudy) metrics measure model output effectiveness against expected input (F1-Score), considering word retrieval (Recall) and relevance (Precision) [5].
- **METEOR**: METEOR (Metric for Evaluation of Translation with Explicit Ordering) is a graph-based metric measuring translation effectiveness in NLP by evaluating grammatical and syntactic structure [2].

2.5 Second Fine-Tuning Round

2.5.1 Fine-Tuning Strategy. Between the first and second model fine-tuning rounds, we expanded our GPU capacity from 6GB to 40GB. This allowed us to adopt the more robust T5-Base model with 220 million parameters. The dataset required new treatment to remove unnecessary complexity elements and reduce its size for the robust training configurations. Therefore, the following elements were changed:

- **Title Tags**: During preprocessing, these tags were replaced with "NA", which proved unnecessary.
- **Steps to Reproduce**: The first three reproduction steps showed minimal variations, so we removed them from training.
- **Additional Information**: Though removed in initial processing, blank fields remained, so we excluded them from training.

To measure model performance realistically, a senior software tester classified the model's responses. Responses with hallucinations or not matching input were marked "Failures", while accurate responses without hallucinations were marked "Successes".

2.5.2 Fine-Tuning Execution. The second fine-tuning used "T5-Base" model with 800 bugs (650 for training, 150 for testing). The 10 most common bug types were identified, and the senior tester conducted 30 validations on these types to evaluate model responses and identify issues.

2.6 Third Fine-Tuning Round

2.6.1 Fine-Tuning Strategy. For the final fine-tuning phase, we adopted a strategy where models are trained with small groups of distinct bug types. This approach tested whether model performance was influenced by the amount of bug report information it was learning. A new collection of bugs was categorized by type, resulting in 3 datasets of 800 bugs each. Three iterations of the T5-Base model were prepared for training with these datasets:

- **Alpha Model**: 6 types of device system bugs.
- **Beta Model**: 1 type of server bug.

- **Gamma Model**: 2 types of device configuration bugs.

2.6.2 Fine-Tuning Execution. The third fine-tuning used 3 iterations of "T5-Base" model with 800 bugs (650 for training, 150 for testing). Since there are 3 models, a system was developed to analyze input, determine failure type, and select the appropriate model for processing.

3 Results

First Fine-Tuning Round: As result from the first fine-tuning round, we obtained that the model achieved 11.76% accuracy, results that were considered negative and displayed not only that we must alter our training approach, but the metrics we were using as well.
Second Fine-Tuning Round: After the second fine-tuning, we redid the dataset treatment, used a more complex model with increased processing power and new metrics, achieving improved values of 0.8286 BLEU and 0.8867 METEOR. The simulation included 30 test failures, with 17 unusable and 13 usable responses (57% failures and 43% successes). This failure rate indicates the model remains far from satisfactory due to numerous hallucinations.
Third Fine-Tuning Round: Table 1 shows results after the third fine-tuning round. Specialized models for failure types improved results. Gamma model covering 2 low-complexity bug reports performed best with 0.9634 BLEU and 0.9845 METEOR, while Alpha model covering more reports performed worst with 0.8641 BLEU and 0.9267 METEOR. Beta model covering 1 high-complexity report had reasonable performance. The senior test evaluation showed 70% success, demonstrating specialized models' superiority.

Table 1: Results of the Third Fine-Tuning Round

Model	Results	
T5-Base Alpha	BLEU	0.8641
	METEOR	0.9267
	Precision	0.9467
	Recall	0.9436
	F1-Score	0.9450
T5-Base Beta	BLEU	0.8572
	METEOR	0.9259
	Precision	0.9500
	Recall	0.9427
	F1-Score	0.9463
T5-Base Gamma	BLEU	0.9845
	METEOR	0.9634
	Precision	0.9898
	Recall	0.9886
	F1-Score	0.9892

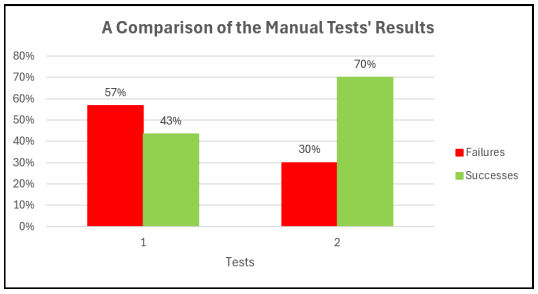

Figure 1: A Comparison of Test Results between 2nd and 3rd Fine-Tuning Stages

4 Conclusions

This study investigated the optimization of bug report writing using LLMs. We conducted a study using real bugs from a software industry testing team to fine-tune the T5 model to generate bug reports based on brief problem descriptions. After the first fine-tuning stage, the accuracy was 0.1176. The second stage achieved a BLEU of 0.8286 and METEOR of 0.8867. However, real-world testing showed 43% success and 57% failure rates, indicating poor practical usability. The third fine-tuning stage, which uses separate datasets and models for similar bug groups, improves the results to 70% success and 30% failure rates. This study demonstrates that automatic bug report generation using LLMs is feasible. Using specialized models for similar bug types proved to be more cost-efficient when the processing power was limited. Future work could explore using more robust T5 models and larger datasets with greater processing powers. We plan to conduct a case study with a test team using the LLM for bug reporting.

ACKNOWLEDGMENTS

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020. The authors extend their gratitude to the Software Engineering Laboratory for its contribution to the IT infrastructure.

REFERENCES

[1] Shengnan An, Zexiong Ma, Zeqi Lin, Nanning Zheng, Jian-Guang Lou, and Weizhu Chen. 2024. Make your llm fully utilize the context. *Advances in Neural Information Processing Systems* 37 (2024), 62160–62188.
[2] Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the acl workshop on intrinsic and extrinsic evaluation measures for machine translation and/or summarization*. 65–72.
[3] Nicolas Bettenburg, Sascha Just, Adrian Schröter, Cathrin Weiss, Rahul Premraj, and Thomas Zimmermann. 2008. What makes a good bug report?. In *Proceedings of the 16th ACM SIGSOFT International Symposium on Foundations of Software Engineering* (Atlanta, Georgia) *(SIGSOFT '08/FSE-16)*. Association for Computing Machinery, New York, NY, USA, 308–318. doi:10.1145/1453101.1453146
[4] Steven Davies and Marc Roper. 2014. What's in a bug report?. In *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement* (Torino, Italy) *(ESEM '14)*. Association for Computing Machinery, New York, NY, USA, Article 26, 10 pages. doi:10.1145/2652524.2652541
[5] Chin-Yew Lin. 2004. Rouge: A package for automatic evaluation of summaries. In *Text summarization branches out*. 74–81.
[6] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 311–318.
[7] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. arXiv:1910.10683 [cs.LG] https://arxiv.org/abs/1910.10683
[8] Vincenzo Scotti, Licia Sbattella, and Roberto Tedesco. 2023. A primer on seq2seq models for generative chatbots. *Comput. Surveys* 56, 3 (2023), 1–58.
[9] Cegu Vima, Hanger Bosch, and John Harringstone. 2024. Enhancing inference efficiency and accuracy in large language models through next-phrase prediction. (2024).
[10] Haiyan Zhao, Hanjie Chen, Fan Yang, Ninghao Liu, Huiqi Deng, Hengyi Cai, Shuaiqiang Wang, Dawei Yin, and Mengnan Du. 2024. Explainability for Large Language Models: A Survey. *ACM Trans. Intell. Syst. Technol.* 15, 2, Article 20 (Feb. 2024), 38 pages. doi:10.1145/3639372
[11] Thomas Zimmermann, Rahul Premraj, Nicolas Bettenburg, Sascha Just, Adrian Schroter, and Cathrin Weiss. 2010. What Makes a Good Bug Report? *IEEE Transactions on Software Engineering* 36, 5 (2010), 618–643. doi:10.1109/TSE.2010.63