Automatic Generation of Test Cases based on Bug Reports: a Feasibility Study with Large Language Models

Laura Plein, Wendkûuni C. Ouédraogo, Jacques Klein, Tegawendé F. Bissyandé
firstname.lastname@uni.lu
University of Luxembourg

ACM Reference Format:
Laura Plein, Wendkûuni C. Ouédraogo, Jacques Klein, Tegawendé F. Bissyandé. 2024. Automatic Generation of Test Cases based on Bug Reports: a Feasibility Study with Large Language Models. In *2024 IEEE/ACM 46th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '24), April 14–20, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3639478.3643119

1 INTRODUCTION

Tests suites are a key ingredient in various software automation tasks. Recently, various studies [4] have demonstrated that they are paramount in the adoption of latest innovations in software engineering, such as automated program repair (APR) [3]. Test suites are unfortunately often too scarce in software development projects. Generally, they are provided for regression testing, while new bugs are discovered by users who then describe them informally in bug reports. In recent literature, a new trend of research in APR has attempted to leverage bug reports in generate-and-validate pipelines for program repair. Even in such cases, when an APR tool generates a patch candidate, if test cases are unavailable, developers must manually validate the patch, leading to a threat to validity.

On the one hand, automatic test generation approaches in the literature [2], unfortunately, either target unit test cases and thus do not cater to the need for revealing complex bugs that users face in the execution of software, or require formally-defined inputs such as the function signatures, or even the test oracle. On the other hand, bug reports are pervasive, but remain under-explored. There is thus a need to investigate the feasibility of test case generation by leveraging bug reports. Our ultimate objective indeed is to address a challenge in the adoption of program repair by practitioners, towards ensuring that patches can be automatically generated and validated for bugs that are reported by users.

Concretely, we observe that, while bug reports can quickly be overwhelming (in terms of high quantity and/or low quality) for developers, they are still recognized to contain a wealth of information. Unfortunately, such information hidden in natural language informality can be difficult to extract, contextualize, and leverage for specifying program executions. Nevertheless, recent advances in Natural Language Processing (NLP) have opened up new possibilities in software engineering. In particular, with the advent of large language models (LLMs), a wide range of tasks have seen machine learning achieve, or even exceed human performance.

In this work, we propose to *study the feasibility of exploiting LLMs towards producing executable test cases based on informal bug reports.* Our experiments build on ChatGPT [1], a general-purpose LLM, and codeGPT [5], a code-specific LLM. The performance of test case generation with LLMs is assessed based on the Defects4J repository which includes real-world faults from various Java software development projects.

2 EXPERIMENTAL SETUP

To train a model that is able to generate the required test case, our pipeline includes the following steps: (1) We start by identifying Java projects included and collecting their bug reports; (2) then, we use LLMs (either an online service or a pre-trained model that we fine-tune) for the purpose of generating test cases; (3) once the test cases are generated, they are appended to the existing test suite of the project to assess their *executability* and *validity* as well as their *relevance* for the associated reported bug; (4) finally, once we generated a relevant test case for the given bug, APR tools can now be applied towards generating and validating the bug-fixing patch. Figure 1 provides an example bug report for which ChatGPT generated a correct test case (Listing 1).

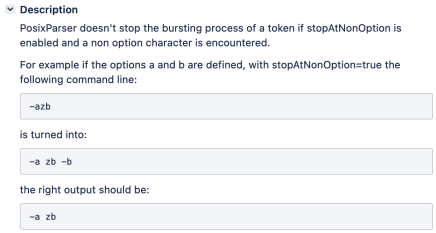

Figure 1: Example bug report from Defects4J dataset - Cli 17

```java
public void testPosixParserStopAtNonOption() throws
     ParseException {
    String[] args = {"-azb"};
    Options options = new Options();
    options.addOption("a", false, "Option A");
    options.addOption("b", false, "Option B");
    CommandLineParser parser = new PosixParser();
    CommandLine cmd = parser.parse(options, args, true);

    assertTrue(cmd.hasOption("a"));
    assertTrue(cmd.getOptionValue("a").isEmpty());
    assertTrue(cmd.getArgs()[0].equals("zb"));
    assertFalse(cmd.hasOption("b"));
}
```

Listing 1: ChatGPT-generated valid test case for Cli 17

ICSE-Companion '24, April 14–20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0502-1/24/04.
https://doi.org/10.1145/3639478.3643119

		single generation attempt					multiple (5) generation attempts				
Project	# of bug reports	overall exec.	overall valid.	valid. over exec.	overall relev.	relev. over exec.	overall exec.	overall valid.	valid over exec.	overall relev.	relev. over exec.
Chart	6	0%	0%	0%	0%	0%	33%	17%	50%	17%	50%
Cli	30	37%	23%	64%	10%	27%	53%	37%	69%	17%	31%
Closure	127	6%	3%	50%	3%	50%	46%	28%	59%	3%	7%
Lang	60	25%	10%	40%	17%	67%	60%	43%	72%	27%	44%
Math	100	13%	3%	23%	3%	23%	43%	15%	35%	6%	14%
Time	19	32%	32%	100%	0%	0%	84%	17%	81%	0%	0%
Total	342	15%	7%	47%	6%	38%	50%	30%	59%	9%	19%

Table 1: Performance of ChatGPT on the task of test case generation based on bug report.
Legend: exec. → executability; valid. → validity. relev. → relevance

2.1 Research Questions

To evaluate the feasibility of translating informal bug reports into executable test cases we investigate the following research questions: ❶ Can informally-written bug reports be translated into executable test cases with ChatGPT? ❷ To what extent can fine-tuned models compare to the baseline results achieved with ChatGPT? ❸ Can LLMs actually generate executable test cases for new bugs?

2.2 Metrics

- **Executability** is a binary metric which describes whether the test case is directly executable on the corresponding project version without any manual changes.
- **Validity:** an executable test case may or may not fail on the target buggy program. We follow the convention of patch validation in program repair and consider the generated test case to be valid only when it, indeed, fails on the buggy program. Otherwise it is considered as invalid.
- **Relevance** is another binary metric which describes if a *valid* test case can not only reproduce the bug but also validate the patched version.

3 EXPERIMENTAL RESULTS

3.1 [RQ1]: LLM baseline performance on test case generation with ChatGPT

Our experimental results in Table 1 show that ChatGPT can be prompted with bug reports to generate executable test cases for 50% of the input samples. Beyond *executability*, about 30% of the bugs could be reproduced with valid test cases, and about 9% of all generated test cases were actually relevant. Nonetheless, we note that over half (59%) of the executable test cases were valid test cases. These results, which are based on an off-the-shelf LLM as-a-service, show promises for automated test case generation, leveraging complex information from user-reported bugs.

3.2 [RQ2]: Performance of CodeGPT, a code-specific and fine-tuned LLM

Fine-tuning CodeGPT yielded an LLM that generates executable test cases for 24% of the bug reports with a single generation attempt (cf. Table 2). This rate is substantially larger than the one achieved by the ChatGPT single generation baseline (15%). However, when performing several generation attempts, ChatGPT achieves a significantly higher rate of success. These results suggest that a fine-tuned LLM could be beneficial in an automated pipeline where a single shot is adequate, whereas ChatGPT would be more useful in recommendation scenarios. The results further suggest that more powerful models should be investigated in future work.

	single generation		5 generations	
	ChatGPT	CodeGPT	ChatGPT	CodeGPT
Executability of all test cases	15%	24%	50%	34%
Validity of all test cases	7%	15%	30%	17%
Validity of executable test cases	47%	60%	59%	51%
Relevance of all test cases	6%	5%	9%	6%
Relevance of executable test cases	38%	20%	19%	17%

Table 2: Performance of fine-tuned CodeGPT vs ChatGPT on the task of bug report driven test case generation

3.3 [RQ3]: Performance on unseen bugs

ChatGPT, an LLM-as-a-service, was proven capable of generating executable test cases for newly reported bugs as shown in Table 3. Overall, in over 55% cases, the new bug reports could effectively serve as prompts to generate an executable test case. Through manual analysis, we confirmed that the generated test case reflects the described behaviour.

Project	# new bugs	# executable	% executable	% valid over exec.
Cli	3	0	0%	-
Lang	12	8	67%	50%
Math	5	2	40%	-
JacksonDatabind	5	4	80%	100%
Jsoup	13	7	54%	-
Total	38	21	55%	47%

Table 3: Executability for newly reported bugs

Acknowledgement. This work is supported by funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation program (grant agreement No. 949014).

REFERENCES

[1] Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems* 33 (2020), 1877–1901.

[2] Gordon Fraser and Andrea Arcuri. 2011. Evosuite: automatic test suite generation for object-oriented software. In *Proceedings of the 19th ACM SIGSOFT symposium and the 13th European conference on Foundations of software engineering*. 416–419.

[3] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. *Commun. ACM* 62, 12 (2019), 56–65.

[4] Kui Liu, Anil Koyuncu, Tegawendé F Bissyandé, Dongsun Kim, Jacques Klein, and Yves Le Traon. 2019. You cannot fix what you cannot find! an investigation of fault localization bias in benchmarking automated program repair systems. In *2019 12th IEEE conference on software testing, validation and verification (ICST)*. IEEE, 102–113.

[5] Shuai Lu, Daya Guo, Shuo Ren, Junjie Huang, Alexey Svyatkovskiy, Ambrosio Blanco, Colin B. Clement, Dawn Drain, Daxin Jiang, Duyu Tang, Ge Li, Lidong Zhou, Linjun Shou, Long Zhou, Michele Tufano, Ming Gong, Ming Zhou, Nan Duan, Neel Sundaresan, Shao Kun Deng, Shengyu Fu, and Shujie Liu. 2021. CodeXGLUE: A Machine Learning Benchmark Dataset for Code Understanding and Generation. *CoRR* abs/2102.04664 (2021).