An Empirical Comparison of LLMs on C/C++ Program Repair

Professor: Vladimir Filkov

TAs: Arefeh Yavary and Ty Feng

Working Group 3: Musheng He

Xinzhuo Hu

Rishika Garg

Srikkanth Ramachandran

Harshil Bhandari

UCDAVIS
UNIVERSITY OF CALIFORNIA

January 24, 2024

1 Introduction

Automated program repair (APR) plays a crucial role in the software development lifecycle by automating the correction of erroneous code in programs and reducing human intervention. Current state-of-the-art template-based APR tools [1, 2] and numerous Neural Machine Translation(NMT)-based techniques [3, 4] have limitations regarding the types and quantity of bugs they can effectively fix.

Recent advancements have seen researchers directly leveraging Large Language Models (LLMs) for APR [5, 6, 7, 8]. AlphaRePair [6] is the first infilling-style LLM-based APR using CodeBERT to generate fixed code lines directly. Xia et al. [7] explored the application of larger LLMs and different LLM architectures for APR. Huang et al. [8] conducted a study on the repair capability of large language models of code (LLMCs) in the fine-tuning paradigm. Jin et al. [9] proposed a Retriever-Generator APR framework InferFix based on a powerful LLM (finetuned 12 billion parameter Codex model). However, these studies rely on fine-tuned LLMs, which require additional model training and specific training datasets. Also, existing works predominantly evaluate models on APR benchmarks of Java and Python, revealing a lack of state-of-the-art APR tools tailored for real-world C/C++ software development [10].

Our goal for this project is to **conduct a comprehensive study on the program repair capabilities of 3 general-purpose LLMs concerning real-world C/C++ programs.** Our evaluation will encompass multiple dimensions, such as correct fixes, repair accuracy, and response time. Additionally, we aim to implement a fully-automated conversation-driven approach [11] to quantitatively assess LLMs' true repair capabilities across various repair scenarios.

2 Research Questions

Theory: Our thesis is that if LLMs can produce code snippets effectively, then they might also be effective code debuggers. Modern software can exceed millions of lines and an automated tool for detecting bugs has been an active research area for a long time. Our goal in this section is to investigate evidence of the following hypothesis.

Hypothesis: LLMs can be used as an effective automated tool for software debugging.

Towards verifying the hypothesis, we form the following research questions:

RQ 1 (Effectiveness). How effective are LLMs in identifying and fixing different types of bugs in C/C++ and does their effectiveness vary under different repair scenarios?

RQ 2 (Chain of Thought). Do LLMs perform significantly better with additional context information in extra rounds of interactions with the user?

RQ 3 (Competitiveness). How do LLMs compare with existing traditional APR tools?

2.1 Research philosophy

In formulating our research questions, we follow the deductive method of reasoning.

RQ 1 and 2 are exploratory, descriptive, and classification questions designed to figure out properties of LLM that have not been extensively studied in literature. RQ 3 is a comparative-descriptive question comparing different public LLMs against a traditional APR tool.

Our overall philosophy has been towards eclectic/pragmatists. We focus on obtaining practical data and providing evidence to the software community about using new technology (public pre-trained LLMs) to promote software development.

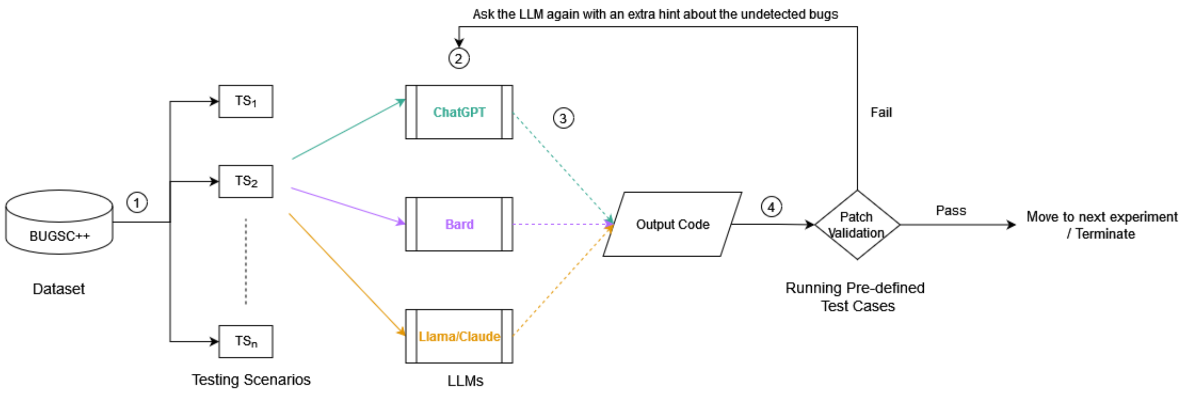
Ask the LLM again with an extra hint about the undetected bugs
2
Fail
TS1
ChatGPT
3
TS2
1
Output Code
4
Patch Validation
Pass
Move to next experiment / Terminate
BUGSC++
Bard
Dataset
Running Pre-defined Test Cases
TSn
Llama/Claude
Testing Scenarios
LLMs

Figure 1: Workflow of the project

3 Approaches

3.1 Data

In evaluation, our research employs *BUGSC++* [10] as the benchmark to gauge the efficacy of our three experimental LLMs in repairing practical C/C++ programs.

BUGSC++: a new benchmark that contains 215 real-world bugs collected from 22 open-source C/C++ projects containing pairs of buggy and patch versions of the source project. It provides CLIs to checkout, build, and test individual bugs in a Docker environment that ensures reproducibility. Moreover, *BUGSC++* applies 4 tag categories including a total of 14 bug types to distinguish different defects.

Our research centers on the repair setting that necessitates fixing bugs, security vulnerabilities, and memory errors. Consequently, before giving LLMs the initial prompts structured with buggy functions from *BUGSC++*, it is imperative to prepare these functions with corresponding information (e.g., error message, error line number) under multiple testing scenarios.

3.2 Process and Potential Issues

Our research aims to evaluate the efficacy of LLMs in real-world C/C++ software development. Figure 1 shows the workflow of our project. For task 1, we plan to classify the buggy functions into 6 testing scenarios, considering 3 different bug types and single/multiple error lines. For task 3 we extract patch candidates outputted by LLMs and conduct patch validation in task 4 (RQ1). We also plan to examine the improvements of LLMs by prompting additional information in extra conversation rounds as task 2 (RQ2). Furthermore, we intend to employ a traditional APR tool [12] on our data, establishing it as a baseline for comparison (RQ3).

Potential issues may arise in plausible patches generated by LLMs that can pass the corresponding test suite. However, plausible patches may not always be correct since the test suite can be incomplete. To address this concern, our strategy involves utilizing manual checks and the repair accuracy metric, following prior research [6, 8]. Another potential challenge lies in the baseline model. Many traditional APR tools designed for repairing C programs have become outdated and are no longer maintained, posing a risk of failure when relying on such tools.

4 Timeline

Stage 1: Project Setup (Duration: 1 week)
Immediate Goal: Deploy the BUGSC++ dataset and integrate LLMs' APIs into the project.
Achievements: Successfully set up the project and integrate all necessary components, ensuring basic functional compatibility.

Stage 2: Data Preprocessing and Prompt Automation (Duration: 2 weeks)
Immediate Goal: Prepare 6 testing scenarios (3 bug types with single/multiple error lines) and a fully automated script to perform LLM query on program repairing.
Achievements: Achieve the full automation on LLM query, output analysis, patch extract, and patch validation using scripts calling 3 LLMs' APIs.

Stage 3: RQ2 and RQ3 Study (Duration: 2 weeks)
Immediate Goal: Implement automation framework to provide LLM with extra information of failed patches until obtaining a correct patch or reaching maximum conversation turns.
Achievements: Complete all patch generation experiments on 3 LLM models using this conversation-driven technique. Evaluate the data on a traditional APR tool [12] as our baseline.

Stage 4: Evaluation and Documentation (Duration: 1 week)
Immediate Goal: Collect all experimental results, perform evaluation, and discussion.
Achievements: Completion of a comprehensive evaluation of our research, Obtain the results with validity and reliability accompanied by a final report.

5 Team Membership & Attestation

	Introduction	Research Questions	Approaches - Data	Approaches - Process, Issues, Workflow	Timeline
Musheng He			√		√
Xinzhuo Hu	√			√	√
Rishika Garg				√	
Srikkanth Ramachandran		√			
Harshil Bhandari				√	

References

[1] Samuel Benton, Xia Li, Yiling Lou, and Lingming Zhang. On the effectiveness of unified debugging: An extensive study on 16 program repair systems. In *2020 35th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 907–918, 2020.

[2] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F. Bissyandé. Tbar: revisiting template-based automated program repair. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2019, page 31–42, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. A syntax-guided edit decoder for neural program repair. In *Proceedings of the 29th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2021, page 341–353, New York, NY, USA, 2021. Association for Computing Machinery.

[4] Nan Jiang, Thibaud Lutellier, and Lin Tan. Cure: Code-aware neural machine translation for automatic program repair. In *Proceedings of the 43rd International Conference on Software Engineering*, ICSE '21, page 1161–1173. IEEE Press, 2021.

[5] Julian Aron Prenner, Hlib Babii, and Romain Robbes. Can openai's codex fix bugs? an evaluation on quixbugs. In *Proceedings of the Third International Workshop on Automated Program Repair*, APR '22, page 69–75, New York, NY, USA, 2022. Association for Computing Machinery.

[6] Chunqiu Steven Xia and Lingming Zhang. Less training, more repairing please: revisiting automated program repair via zero-shot learning. In *Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2022, page 959–971, New York, NY, USA, 2022. Association for Computing Machinery.

[7] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. Automated program repair in the era of large pre-trained language models. In *Proceedings of the 45th International Conference on Software Engineering*, ICSE '23, page 1482–1494. IEEE Press, 2023.

[8] Kai Huang, Xiangxin Meng, Jian Zhang, Yang Liu, Wenjie Wang, Shuhao Li, and Yuqing Zhang. An empirical study on fine-tuning large language models of code for automated program repair. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 1162–1174, 2023.

[9] Matthew Jin, Syed Shahriar, Michele Tufano, Xin Shi, Shuai Lu, Neel Sundaresan, and Alexey Svyatkovskiy. Inferfix: End-to-end program repair with llms. In *Proceedings of the 31st ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ESEC/FSE 2023, page 1646–1656, New York, NY, USA, 2023. Association for Computing Machinery.

[10] Gabin An, Minhyuk Kwon, Kyunghwa Choi, Jooyong Yi, and Shin Yoo. Bugsc++: A highly usable real world defect benchmark for c/c++. In *2023 38th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, pages 2034–2037, 2023.

[11] Chunqiu Steven Xia and Lingming Zhang. Keep the conversation going: Fixing 162 out of 337 bugs for $0.42 each using chatgpt. 2023.

[12] Claire Le Goues, ThanhVu Nguyen, Stephanie Forrest, and Westley Weimer. Genprog: A generic method for automatic software repair. *IEEE Transactions on Software Engineering*, 38(1):54–72, 2012.