RepairCAT: Applying Large Language Model to Fix Bugs in AI-Generated Programs

Nan Jiang
jiang719@purdue.edu
Purdue University
West Lafayette, USA

Yi Wu
wu1827@purdue.edu
Purdue University
West Lafayette, USA

ABSTRACT

Automated program repair has been a crucial and popular domain for years, and with the development of large language models (LLMs) and the trend of using LLMs for code generation, there comes the new challenge of fixing bugs in LLM-generated (AI-generated) programs. In this work, we introduce RepairCAT, a simple and neat framework for fine-tuning large language models for automated repairing Python programs. Our experiments built on StarCoder-1B successfully generated patches fixing the failed test cases for 14 out of 100 bugs in the Python programs, 2 of which passed all the public test cases and were considered plausible.

KEYWORDS

Automated Program Repair, Large Language Model

ACM Reference Format:
Nan Jiang and Yi Wu. 2024. RepairCAT: Applying Large Language Model to Fix Bugs in AI-Generated Programs. In *2024 ACM/IEEE International Workshop on Automated Program Repair (APR '24), April 20, 2024, Lisbon, Portugal*. ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3643788.3648020

1 INTRODUCTION

Automated program repair (APR) [5, 17, 18] helps developers improve software reliability by generating patches automatically to repair software defects, which has become a crucial domain recently. A standard setup of an APR framework takes the buggy program and optionally the bug location as input, and aims to provide a list of candidate patches that can be used to replace the buggy code in the program.

With the impressive capability of large language models (LLMs) on software engineering domains, applying LLMs for code generation has been widely explored, which raises new challenges of detecting and repairing the potential bugs in AI-generated code, i.e., APR for AI-generated programs. Although similar to traditional APR that aims to fixing bugs in developers code, bugs in AI-generated code may have different patterns [4, 13].

Much deep learning (DL)-based APR techniques [1–3, 9, 10, 12, 15, 24, 25, 27] adapt DL models to take a buggy software program as input and generate a patched program. With the strong learning capability of DL models, these techniques learn diverse patterns of transforming buggy programs to patched programs from large code corpora, and many [10, 25, 27] outperform traditional template-based [7, 14], heuristic-based [19, 20, 26] and "a photo of pedestrians"constraint-based [16, 22, 23] APR techniques. And recently, LLM themselves have also been explored for APR [8, 21], with the general knowledge (including natural language and programming) learned from pre-training, LLM-based APR techniques quickly outperforming DL-based techniques building from the scratch significantly.

To apply LLM-based APR techniques for repairing AI-generated programs, the main challenge to overcome is the lack of data. Different from the program repair data collected from open-source platforms such as GitHub, AI-generated bug data is much less in the wild. Besides, AI-generated code contains special bugs that are different from human developers [4], such as using improper identifier names indicating the wrong algorithm, introducing irrelevant helper functions, and so on. Such special patterns of AI-generated code can not be easily learned from human developer-written code. Thus, we propose a two-stage framework: (1) we fine-tune a code generation LLM using a code generation dataset in a certain domain, which results in a code generation model that can be used to create a large AI-generated bug dataset, and (2) in the second stage, we use the created AI-generated bug data to fine-tune further a program repair model which can successfully fix bugs written by code generation models.

2 APPROACH

This section introduces our approaches, especially the two-stage framework of creating an AI-generated bug dataset and fine-tuning APR models.

2.1 Fault Localization

Fault localization has been a difficult part of program repair for a long time, and in practice, the precision of which is still not good enough and causes the deduction of the program repair framework's performance. In our framework, we decided not to perform fault localization particularly, instead, the whole buggy program is treated together and let the model decide where to fix. The input to the tool is the whole buggy code file, and the output of the tool is the complete fixed code file.

2.2 Patch Generation

First, we need to collect AI-generated Python bugs as our training data. We leverage StarCoder-1B [11] to generate buggy code. Since the APR competition provides the whole Python buggy class as

APR '24 , April 20, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0577-9/24/04
https://doi.org/10.1145/3643788.3648020

input, we fine-tuned StarCoder-1B to generate Python class with APPS dataset [6] as shown in Figure 1 . The APPS dataset consists of 10,000 Python problems with natural language question descriptions, corresponding solutions, and test cases. We clean the APPS dataset by only keeping the data whose solution starts with "`import ... class Solution`" that follows the APR competition example input format. Then we use the fine-tuned StarCoder-1B to generate 10 solutions for each question. We run test cases to filter and keep the incorrect solutions generated by the model. This results in 4,447 incorrect solutions that serve as the training data for APR.

In the second stage, we use the buggy code collected from the first stage as the input to the model and train the model to generate the correct code. The input to the model is the question description followed by the buggy code and a single line of comment"`# Please fix the above code:`". Figure 2 shows an example of APR fine-tuning data, whose input is the question description and the sampled buggy code by the code generation LLM, and the output is the fixed code.

```
Given an integer array sorted in non-decreasing order, there
is exactly one integer in the array that occurs more than
25% of the time. Return that integer.

Example 1:
Input: arr = [1,2,2,6,6,6,6,7,10]
Output: 6

Constraints:
1 <= arr.length <= 10^4
0 <= arr[i] <= 10^5
```

(a) Input: Question descrisption

```
class Solution:
    def findSpecialInteger(self, arr: List[int]) -> int:
        len_arr = len(arr)
        for n in set(arr):
            if arr.count(n) > len_arr * 0.25:
        return n
```

(b) Output: Python class as solution

Figure 1: Fine-tuning data from APPS dataset.

3 EXPERIMENTAL SETUP

In the first stage, we fine-tuned StarCoder-1B using APPS dataset with 1 epoch and a batch size of 4. In the second stage, We fine-tuned StarCoder-1B with the collected training data with 3 epochs and a batch size of 2. We use Adam optimizer with learning rate of $5e^{-5}$ and a weight decay of 0.01 for both stages.

4 RESULT

We make our framework and tool participate in the AI-generated code track at the 1st International Competition on Automated Program Repair [1]. During the competition, each bug is given one hour for patch generation and validation. We let the fine-tuned StarCoder-1B model sample at most 1,000 candidate patches using a temperature of 0.8. The generated candidate patches are applied

[1]https://apr-comp.github.io/index.html

```
"""Python
Suppose an array sorted in ascending order is rotated at
some pivot unknown to you beforehand.
You are given a target value to search. If found in the
array return true, otherwise return false.

Example 1:
Input: nums = [2,5,6,0,0,1,2], target = 0
Output: true
...
"""

from typing import *
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        start = 0
        end = len(nums) - 1
        while start <= end:
            ...
        return start

# Please fix the above code:
```

(a) Input: Question descrisption and buggy code

```
from typing import *
class Solution:
    def search(self, nums: List[int], target: int) -> int:
        left = 0
        right = len(nums) - 1
        while left <= right:
            ...
        return -1
```

(b) Output: Fixed code

Figure 2: Fine-tuning data sampled by the code generation LLM.

and run the test cases for validation. We only report the patches that pass the failing test cases to reduce false positives.

Our tool successfully generates patches that pass the failing test cases of 14 bugs, 2 of which also pass all the public test cases and are considered plausible, outperforming the baseline approaches.

5 AVAILABILITY

According to the rules of the competition, we open-source our tool as follows.

5.1 DOI of Artifact

The source code to train the LLMs and build the tools is archived in a Zenodo link: https://zenodo.org/record/8429188, and is also accessible on GitHub at https://github.com/jiang719/cerberus-repaircat/tree/ai-generated-code.

5.2 Docker Image

The docker image including the tools and trained models is uploaded as a docker image. To pull the pre-built docker image, run: `docker pull jiang719/repaircat-autocode-python`

The hash of the final tool image is `sha256:e6e990b4310e22b3465f5f709cf8f8d38bdcd962df9347f8cb7c91f5c765471e`. Using the docker image and the image hash, one can access our tools in the official competition framework [2].

[2]https://github.com/nus-apr/cerberus

REFERENCES

[1] Saikat Chakraborty, Yangruibo Ding, Miltiadis Allamanis, and Baishakhi Ray. 2022. CODIT: Code Editing With Tree-Based Neural Models. *IEEE Transactions on Software Engineering* 48, 4 (2022), 1385–1399. https://doi.org/10.1109/TSE.2020.3020502

[2] Zimin Chen, Steve Kommrusch, and Martin Monperrus. 2022. Neural Transfer Learning for Repairing Security Vulnerabilities in C Code. *IEEE Transactions on Software Engineering* (2022). https://doi.org/10.1109/TSE.2022.3147265

[3] Zimin Chen, Steve Kommrusch, Michele Tufano, Louis-Noël Pouchet, Denys Poshyvanyk, and Martin Monperrus. 2019. SequenceR: Sequence-to-Sequence Learning for End-to-End Program Repair. *TSE* (2019).

[4] Zhiyu Fan, Xiang Gao, Martin Mirchev, Abhik Roychoudhury, and Shin Hwei Tan. 2023. Automated Repair of Programs from Large Language Models. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1469–1481. https://doi.org/10.1109/ICSE48619.2023.00128

[5] Claire Le Goues, Michael Pradel, and Abhik Roychoudhury. 2019. Automated program repair. *Commun. ACM* 62, 12 (2019), 56–65. https://doi.org/10.1145/3318162

[6] Dan Hendrycks, Steven Basart, Saurav Kadavath, Mantas Mazeika, Akul Arora, Ethan Guo, Collin Burns, Samir Puranik, Horace He, Dawn Song, et al. 2021. Measuring coding challenge competence with apps. *arXiv preprint arXiv:2105.09938* (2021).

[7] Jiajun Jiang, Yingfei Xiong, Hongyu Zhang, Qing Gao, and Xiangqun Chen. 2018. Shaping Program Repair Space with Existing Patches and Similar Code. In *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2018)*. Association for Computing Machinery, New York, NY, USA, 298–309. https://doi.org/10.1145/3213846.3213871

[8] Nan Jiang, Kevin Liu, Thibaud Lutellier, and Lin Tan. 2023. Impact of Code Language Models on Automated Program Repair. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1430–1442. https://doi.org/10.1109/ICSE48619.2023.00125

[9] Nan Jiang, Thibaud Lutellier, Yiling Lou, Lin Tan, Dan Goldwasser, and Xiangyu Zhang. 2023. KNOD: Domain Knowledge Distilled Tree Decoder for Automated Program Repair. In *Proceedings of the International Conference on Software Engineering*.

[10] Nan Jiang, Thibaud Lutellier, and Lin Tan. 2021. CURE: Code-Aware Neural Machine Translation for Automatic Program Repair. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. 1161–1173. https://doi.org/10.1109/ICSE43902.2021.00107

[11] Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023. StarCoder: may the source be with you! arXiv:2305.06161 [cs.CL]

[12] Yi Li, Shaohua Wang, and Tien N. Nguyen. 2020. DLFix: Context-Based Code Transformation Learning for Automated Program Repair. In *ICSE* (Seoul, South Korea). ACM, 602–614.

[13] Zongjie Li, Chaozheng Wang, Zhibo Liu, Haoxuan Wang, Dong Chen, Shuai Wang, and Cuiyun Gao. 2023. CCTest: Testing and Repairing Code Completion Systems. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1238–1250. https://doi.org/10.1109/ICSE48619.2023.00110

[14] Kui Liu, Anil Koyuncu, Dongsun Kim, and Tegawendé F Bissyandé. 2019. TBar: Revisiting Template-Based Automated Program Repair. In *ISSTA*. ACM.

[15] Thibaud Lutellier, Hung Viet Pham, Lawrence Pang, Yitong Li, Moshi Wei, and Lin Tan. 2020. CoCoNuT: Combining Context-Aware Neural Translation Models Using Ensemble for Program Repair. In *ISSTA* (Virtual Event, USA). ACM, 101–114.

[16] Matias Martinez and Martin Monperrus. 2018. Ultra-Large Repair Search Space with Automatically Mined Templates: The Cardumen Mode of Astor. In *Search-Based Software Engineering - 10th International Symposium, SSBSE 2018, Montpellier, France, September 8-9, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 11036)*, Thelma Elita Colanzi and Phil McMinn (Eds.). Springer, 65–86. https://doi.org/10.1007/978-3-319-99241-9_3

[17] Martin Monperrus. 2018. Automatic Software Repair: A Bibliography. *ACM Comput. Surv.* 51, 1 (2018), 17:1–17:24. https://doi.org/10.1145/3105906

[18] Martin Monperrus. 2020. The living review on automated program repair. (2020).

[19] Ripon K. Saha, Yingjun Lyu, Hiroaki Yoshida, and Mukul R. Prasad. 2017. ELIXIR: effective object oriented program repair. In *Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering, ASE 2017, Urbana, IL, USA, October 30 - November 03, 2017*, Grigore Rosu, Massimiliano Di Penta, and Tien N. Nguyen (Eds.). IEEE Computer Society, 648–659. https://doi.org/10.1109/ASE.2017.8115675

[20] Ming Wen, Junjie Chen, Rongxin Wu, Dan Hao, and Shing-Chi Cheung. 2018. Context-aware patch generation for better automated program repair. In *Proceedings of the 40th International Conference on Software Engineering, ICSE 2018, Gothenburg, Sweden, May 27 - June 03, 2018*, Michel Chaudron, Ivica Crnkovic, Marsha Chechik, and Mark Harman (Eds.). ACM, 1–11. https://doi.org/10.1145/3180155.3180233

[21] Chunqiu Steven Xia, Yuxiang Wei, and Lingming Zhang. 2023. Automated Program Repair in the Era of Large Pre-Trained Language Models. In *Proceedings of the 45th International Conference on Software Engineering* (Melbourne, Victoria, Australia) *(ICSE '23)*. IEEE Press, 1482–1494. https://doi.org/10.1109/ICSE48619.2023.00129

[22] Yingfei Xiong, Jie Wang, Runfa Yan, Jiachen Zhang, Shi Han, Gang Huang, and Lu Zhang. 2017. Precise Condition Synthesis for Program Repair. In *2017 IEEE/ACM 39th International Conference on Software Engineering (ICSE)*. 416–426. https://doi.org/10.1109/ICSE.2017.45

[23] Jifeng Xuan, Matias Martinez, Favio Demarco, Maxime Clement, Sebastian R. Lamelas Marcote, Thomas Durieux, Daniel Le Berre, and Martin Monperrus. 2017. Nopol: Automatic Repair of Conditional Statement Bugs in Java Programs. *IEEE Trans. Software Eng.* 43, 1 (2017), 34–55. https://doi.org/10.1109/TSE.2016.2560811

[24] He Ye, Matias Martinez, Xiapu Luo, Tao Zhang, and Martin Monperrus. 2022. SelfAPR: Self-supervised Program Repair with Test Execution Diagnostics. In *Proceedings of ASE*. http://arxiv.org/pdf/2203.12755

[25] He Ye, Matias Martinez, and Martin Monperrus. 2022. Neural Program Repair with Execution-based Backpropagation. In *Proceedings of the International Conference on Software Engineering*. https://doi.org/10.1145/3510003.3510222

[26] Yuan Yuan and Wolfgang Banzhaf. 2020. ARJA: Automated Repair of Java Programs via Multi-Objective Genetic Programming. *IEEE Trans. Software Eng.* 46, 10 (2020), 1040–1067. https://doi.org/10.1109/TSE.2018.2874648

[27] Qihao Zhu, Zeyu Sun, Yuan-an Xiao, Wenjie Zhang, Kang Yuan, Yingfei Xiong, and Lu Zhang. 2021. 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*. Association for Computing Machinery, New York, NY, USA, 341–353. https://doi.org/10.1145/3468264.3468544