Reproducing Automatic Program Repair with Codex's paper results

Bardia Mohammadi

1 Introduction

With a large code corpus, OpenAI's Codex model is similar to GPT-3 in that it generates code snippets that are valid on both a syntactical and semantic level based on a brief description provided by the user. As part of the automated software repair field, the goal of this study is to determine if Codex is capable of localizing and fixing bugs. There were 40 bugs in both the Python and Java QuixBugs benchmarks that the article applied as part of its initial evaluation. Although Codex has not been trained in APR, the results of the study indicate that it is effective and competitive with the current state-of-the-art methods of APR, even though it has not been trained in it. The reproduced results show that Codex is more effective at repairing Java than the results reported in the article.

2 Reproducing methods

The experiments in the article rely heavily on QuixBugs, a benchmark of 40 buggy algorithm implementations, including their correct versions and test cases, in order to conduct all of the experiments. The methodology section of the article provides a description of the benchmark.

As Codex can only be interacted with via the prompt that is provided, it is essential that the authors experiment with a variety of input configurations. At the moment, the configuration used in the task is code only; simply type in the buggy function (without any docstring).

```
/// fix the bug in the following method
<buggy code here>
/// fixed method
```

This experiment was conducted using the same configuration as that which was mentioned in the article that was used in the original experiment.

In order to evaluate the results, I held the same constraints and procedures as in the production part, and made sure that all output was checked with Quixbug's unit tests as well as other edge cases.

Algorithm	*Correct*
bitcount	✓
breadth-first-search	✓
bucketsort	✓
depth-first-search	✓
detect-cycle	✓
find-first-in-sorted	✓
find-in-sorted	✓
flatten	✓
gcd	✓
get_factors	✓
hanoi	✓
is-validparenthesization	✓
kheapsort	✓
knapsack	✓
kth	✓
lcs-length	
levenshtein	✓
lis	✓
longest-commonsubsequence	✓
max-sublist-sum	✓
mergesort	
minimum-spanningtree	
next-palindrome	
next-permutation	✓
pascal	✓
possible-change	
powerset	
quicksort	
reverse-linked-list	✓
rpn-eval	✓
shortest-path-length	
shortest-pathlengths	✓
shortest-paths	
shunting-yard	✓
sieve	✓
sqrt	✓
subsequences	✓
to-base	✓
topologicalordering	✓
warp	

Table 1: Reproduced results

3 Obstacles

The main obstacle I had to overcome when producing the result was that Iranians are prohibited from using the OpenAI systems. Despite the fact that I have tried to contact OpenAI support, I have not received any response from them. I struggled a lot with different tools to change my IP address since not all of them are compatible with Iran's network.

4 Other Applications

As part of the code review process, OpenAI Codex may be able to assist. As examples of how it can be helpful, here are some ways in which it can be helpful:

1. Code suggestions: The OpenAI Codex is able to offer suggestions for improvements to the code, such as more efficient algorithms, better variable names, or pointing out potential bugs that need to be corrected.

2. Automated testing: OpenAI Codex can assist in the testing process by automatically generating test cases or by providing feedback on the effectiveness of existing test cases in order to facilitate the testing process.

3. Code formatting: The OpenAI Codex allows code to be formatted automatically according to the coding standards, thereby making it easier for people to read and maintain the code.

4. Security analysis: OpenAI Codex provides code analysis services and can detect potential security vulnerabilities in code, such as SQL injection and buffer overflow attacks.

In the context of code refactoring, OpenAI Codex can be used as it can generate and suggest changes based on input code and requirements, which can lead to the refactoring of the code.

Refactoring is the process of making changes to an existing codebase to enhance its readability, maintainability, and performance, without altering the functionality of the codebase itself. Using OpenAI Codex, we can examine the existing code and suggest changes that will improve its structure, remove redundancies, and make it more efficient by analyzing it and suggesting the necessary changes.

Using OpenAI Codex, for instance, if you have a large code base with several repeated code blocks, you will learn how to refactor that code by identifying functions or classes that can be reused throughout the codebase, thereby reducing redundancy and improving readability of the code.

It is important to keep in mind that while OpenAI Codex can provide helpful suggestions for refactoring and reviewing code, it is ultimately up to the developer to decide whether or not the suggested changes are appropriate for the specific use case they are dealing with as well as to thoroughly test the

refactored code to ensure that it still works as intended after it has been refactored. OpenAI Codex is still a tool to help streamline and improve the code review process, but it should not be viewed as a replacement for human expertise. Human review is still essential in order for code quality, maintainability, and security to be ensured.

5 Reason for better result

OpenAI Codex's improved performance may be due to a number of factors. There were billions of lines of code from a variety of programming languages and domains included in the training dataset for OpenAI Codex compared to previous models. Hence, the model understands programming concepts and syntax better than previous versions. Further, OpenAI has fine-tuned the model with the help of the user's feedback, prompts, and completions, resulting in a more accurate and efficient version of the model. Through iterative training, OpenAI continuously updates and refines the model over time as it is fine-tuned for a specific task and as it is exposed to more data.