How Effective is GPT-4o at Generating Test Assertions?

Author: Adomas Bagdonas (abagdonas@tudelft.nl)

Supervisor: Mitchell Olsthoorn
Responsible professor: Annibale Panichella

1. Background

Software testing is crucial but requires a lot of time and effort. It heavily relies on the quality of the assertions.

Search-Based Software Testing (SBST)

- Research field that focuses on automating test creation
- **Limitations:** poor test case readability and inability to distinguish correct program behavior from incorrect

Large Language Models (LLMs)

- Great at working with natural language
- There is already some research on their abilities to generate tests
- **Limitations:** non-stochastic and prone to code hallucinations

Mutation testing

- One of the most insightful strategies for evaluating the quality of a test
- Creates mutants by modifying the code under test and then checks if any of the assertions detect them

2. Approach

We identified two possible approaches:
1. Static – prompting the LLM once
2. Dynamic – asking the model to further improve the results with more prompts.

We focus only on the static approach.

For each of the classes, GPT-4o generated tests. The results were compared with EvoSuite, a SBST tool.

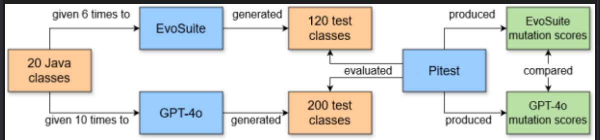

Figure 1: A schematic of our approach

Some of the tests were failing. Thus, we opted for two rounds of comparison:
1. After removing the failing assertions
2. After fixing every assertion manually

To measure the quality of the tests, mutation score was used.

3. Study Design

The research question is: how effective is GPT-4o at generating test assertions with regards to mutation score?

For statistical comparison with EvoSuite we used Wilcoxon rank-sum test together with Vargha-Delaney effect size.

For evaluation we picked 20 Java classes from the SF110 using the following criteria:

1. Must not depend on more than one other class within the project.
2. Must have a cyclomatic complexity of at least 5.
3. The code should contain more complicated logic than basic getters and setters.

Afterwards:
- For each of the classes we generated 10 test classes.
- Then we used Pitest to evaluate their mutation score.
- Compared these scores with the ones obtained by running EvoSuite 6 times.

5. Conclusions and Future Work

Conclusions:
1. Our approach performed slightly worse than EvoSuite in terms of mutation score
2. It improved upon some of the weaknesses of SBST
3. Lastly, GPT-4o is a viable option for developers to use for testing

In the future:
- Analyse different LLMs and languages
- Verify the findings on different datasets
- Try out the dynamic approach

Figure 3: A method with a detected bug

```
public static boolean matchTemplateEnd(String text)
{
    return text != null &&
(text.indexOf("@template_end") != -1)
         || (text.indexOf("@tend") != -1);
}
```

4. Results

200 total generated classes:
- 38 build errors
- 1580 tests after removing every failing assertion
- In total, 71% of the mutants killed
- In terms of readability, the tests seemed to have human-like style

Out of 20 classes, GPT-4o performed significantly ($p <= 0.05$) better in 3 of the cases. Nevertheless, EvoSuite outperformed it in 9 of the cases.

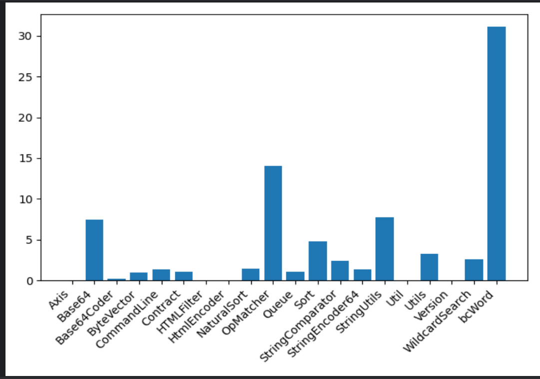

Figure 2: Improvements in mutation score

Manually fixing the failing assertions:
- In total, 225 test methods rewritten
- The mutation score increased to 75%
- Mean mutation score was 81.3% (only 0.08% smaller than EvoSuite)

EvoSuite still had statistically better results in 5 of the cases compared to the 3 cases where GPT-4o surpassed it.

Figure 2 shows the amount of improvement in mutation score after fixing the tests.

Interestingly, a small fraction of the assertions that were failing appeared to be correct. Deeper examinations revealed that some of the assertions managed to detect bugs in the source code.

For example, Figure 3 contains a method from one of the classes. Calling it with a null value causes an exception to be thrown. GPT-4o managed to find the bug in 4 out of 10 test suites for this class.