Assertion Impossible: Evaluating LLMs for Generating Test Assertions

An Empirical Study of Assertion Generation Strategies for LLM-Based Test Oracles

TU Delft

Author: Vanesa Mitseva

Email: v.p.mitseva-1@student.tudelft.nl

Supervisors: Annibale Panichella, Mitchell Olsthoorn

01. BACKGROUND

- ~15.8% of engineers' time is spent on testing → engineers want to automate assertion generation
- LLMs generate compilable assertions, but quality depends on prompting and context
- Fine-tuned models perform better
- **Key knowledge gap:** The generation strategies have not been studied well

02. RESEARCH QUESTIONS

- **RQ1**: How do generation strategies affect **assertion correctness**?
- **RQ2:** How do generation strategies affect **fault detection**?
- **RQ3:** How similar are LLM-generated assertions to **developer-written** ones?

03. GENERATION STRATEGIES

S1. Assertion Generation

```
void testAdd() {
    Calculator calculator = new Calculator();
    <ASSERTION_PLACEHOLDER>
}
```

```
void testAdd() {
    Calculator calculator = new Calculator();
    assertEquals(5, calculator.add(2, 3));
}
```

S2. Assertion Augmentation

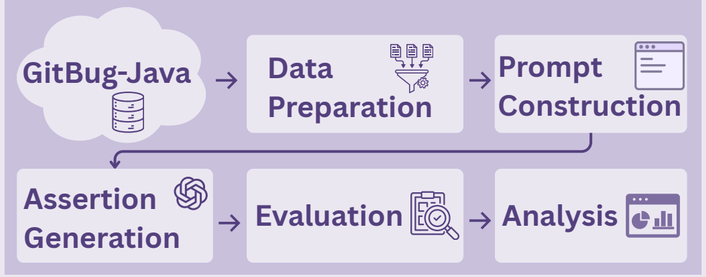
```
void testAdd() {
    Calculator calculator = new Calculator();
    assertEquals(7, calculator.add(3, 4));
}
```

```
void testAdd() {
    Calculator calculator = new Calculator();
    assertEquals(7, calculator.add(3, 4));
    assertEquals(-3, calculator.add(-1, -2));
    assertEquals(0, calculator.add(-1, 1));
}
```

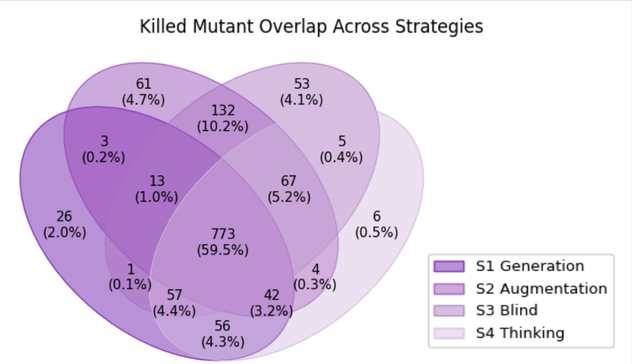
S3. Blind Augmentation: Generating additional assertions without seeing the original ones.

S4. Chain-of-Thought Generation: Step-by-step reasoning about the behaviour of the program before generating.

04. METHODOLOGY



05. RESULTS & CONCLUSIONS



Strategy	Comp. Rate	Exec. Validity	Mut. Score	Bug Det.	Accuracy
Generation	58.3%	70.2%	58.2%	38.5%	36.4%
Augmentation	**74.5%**	**87.7%**	**67.3%**	46.7%	2.7%
Blind	65.6%	77.2%	57.6%	30.8%	4.8%
Thinking	59.7%	75.4%	59.1%	**57.1%**	26.2%

- **RQ1**: *Assertion Augmentation* achieved the highest compilation rate and produced the highest proportion of correctly-executing assertions; *Assertion Generation* performed the worst.
- **RQ2**: *Assertion Augmentation* has the most strategy-exclusive kills *and* produced the highest mutation score, improving the baseline by 1.1%; *CoT* detected the largest number of bugs from GitBug-Java.
- **RQ3**: Similarity to developer assertions is the highest for *Assertion Generation*, and the lowest for *Assertion Augmentation*. This suggests that textual similarity alone is not a reliable measure of assertion quality.
- **General Conclusion**: Based on both RQ1 and RQ2, the two augmentation strategies generally perform better than the generation ones.