Effective test generation using pre-trained Large Language Models and mutation testing

Riddhi Mistry
ID-202201238

Hardi Naik
ID-202201477

***Abstract*—Manual test case generation is time consuming and error prone. This study uses Gemini LLM to generate test cases through zero-shot and few-shot prompting. We measure test quality using mutation testing and improve the test cases with MUTAP. Then, we compare the generated test suites with those from Pynguin on Python programs. MUTAP's improved test cases work better than Pynguin, showing that using LLMs with mutation-based changes gives good results.**

I. Introduction

Test case generation plays a key role in ensuring software quality, yet manual creation is often time-consuming and error-prone. With the rise of Large Language Models (LLMs) like Gemini,Open AI and Lamma there is a growing interest in using these models to automatically generate test cases. In this research, we study how well Gemini can create test cases using two different methods: zero-shot prompting, where the model is given only the function and asked to generate tests without examples, and few-shot prompting, where the model is shown a few examples to guide it.

To check the quality of these test cases, we use mutation testing. This method checks if the test cases can catch small changes (called mutants) made to the original code. We also use a tool called MUTAP, which improves test cases by focusing on mutants that are not caught. Finally, we compare the test cases generated by Gemini to those created by an existing tool called Pynguin. We test all methods on two Python programs and repeat each experiment five times. The result shows that MUTAP-refined test suites perform better than those generated by Pynguin.

II. Conception (Experimental Setup)

A. Goal

To evaluate the effectiveness of test cases generated by Large Language Models (LLMs), specifically Gemini LLM, using zero-shot and few-shot prompting strategies, by assessing their quality through mutation testing with the MUTAP technique.

B. Objectives

- Automatically generate test cases using Gemini with both zero-shot and few-shot prompting.
- Apply mutation testing to evaluate the fault detection capability of the generated test suites.
- Refine the generated test cases using the MUTAP technique.
- Compare the performance of refined test cases against Pynguin generated test suites.
- Measure consistency across multiple runs to validate repeatability and reliability.

C. Research Questions

- RQ1: Can LLMs generate test cases that achieve competitive mutation scores compared to traditional tools like Pynguin?
- RQ2: Does few-shot prompting produce better test cases than zero-shot prompting?
- RQ3: Can the refinement of LLM-generated test cases using MUTAP significantly improve their effectiveness?

D. Frame Hypotheses

Hypothesis 1 (H1): Zero-Shot Setting

Null Hypothesis (H_0):

$$\text{Mutation Score}_{\text{Gemini + MUTAP (Zero-Shot)}} = \text{Mutation Score}_{\text{Pynguin (Zero-Shot)}}$$

Alternative Hypothesis (H_1):

$$\text{Mutation Score}_{\text{Gemini + MUTAP (Zero-Shot)}} > \text{Mutation Score}_{\text{Pynguin (Zero-Shot)}}$$

Hypothesis 2 (H2): Few-Shot Setting

Null Hypothesis (H_0):

$$\text{Mutation Score}_{\text{Gemini + MUTAP (Few-Shot)}} = \text{Mutation Score}_{\text{Pynguin (Few-Shot)}}$$

Alternative Hypothesis (H_1):

$$\text{Mutation Score}_{\text{Gemini + MUTAP (Few-Shot)}} > \text{Mutation Score}_{\text{Pynguin (Few-Shot)}}$$

E. veriables

- **Independent variables :** unit test case generation
- **Treatment** : testcase generation using MUTAP vs testcase generation using Pynguin

- **Dependent variable** : Mutation score
- **Blocking variable** : LLMs, Prompting Technique
- **Experimental Objects** : Python programs
- **Experimetal Subjects** : students

F. Measures:

Mutation Score: Calculated as the ratio of mutants killed to the total number of mutants generated.

G. Experiment Design

TABLE I: One-Factor Block Design: Prompting Technique Sub-columns under MUTAP and Pynguin

Prompt	MUTAP		Pynguin	
	ZS	FS	ZS	FS
p1	✓	✓	✓	✓
p2	✓	✓	✓	✓

H. Experimental process

The experimental workflow begins by using the Gemini Large Language Model (LLM) to automatically generate unit test cases for two Python programs. Two prompting strategies are employed—zero-shot and few-shot—to explore the impact of prompt design on test generation. The resulting test cases are stored separately according to the prompting strategy used.

Mutation testing is then applied to the original Python code using these Gemini-generated test suites to establish baseline Mutation Scores, which indicate the proportion of artificial faults (mutants) that are successfully detected by the tests.

To improve the initial test suite, MuTAP proceeds iteratively. In each iteration, one of the surviving mutants—i.e., a mutant not killed by the current test suite—is selected. The original prompt is then augmented with contextual information about that specific surviving mutant. This enhanced prompt is submitted to the Gemini LLM, which generates additional test cases targeted at that mutant. The newly generated test cases are refined and integrated into the existing test suite. Mutation testing is repeated to evaluate the effectiveness of the refined suite, and the updated Mutation Score is recorded.

In parallel, test suites are generated for the same programs using Pynguin, a search-based automated test generation tool. These Pynguin-generated test suites are also subjected to mutation testing, and their Mutation Scores are computed for comparison.

To ensure statistical reliability and account for the inherent variability in LLM-generated outputs, all steps are repeated five times for each program. The average Mutation Scores across the five runs are computed for each method—Gemini (zero-shot and few-shot) and Pynguin.

All results, including the number of mutants killed and the corresponding Mutation Scores, are systematically documented in structured tables to facilitate quantitative comparison and analysis.

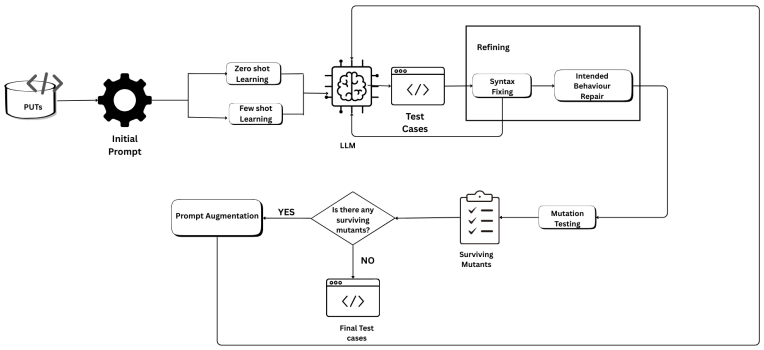

Fig. 1: Flow of the code for generating and evaluating tests using LLMs

III. Execution

A. Preparation

Two Python programs (P1 and P2) were selected for testing purposes. The experimental setup involved configuring the necessary tools: Gemini LLM for test case generation, MuTAP for refining tests based on surviving mutants, and Pynguin for automated test generation. All tools were installed and validated before running the experiments.

B. Execution

Test cases were first generated for both programs using the Gemini LLM with two prompting strategies: zero-shot and few-shot. Mutation testing was then applied using these test suites to obtain the initial Mutation Scores. Next, the generated test cases were improved using MuTAP, which selects surviving mutants and augments the prompt with their context to generate additional, targeted tests using Gemini. The enhanced suite was re-evaluated through mutation testing. Separately, Pynguin was used to generate test cases for the same programs, which were then tested for their Mutation Scores.

C. Data Collection

Mutation scores were collected from each test run. For every test generation strategy and both programs, the average Mutation Score across five runs was calculated. These results

were then used to compare the effectiveness of Gemini with MUTAP and Pynguin in terms of mutation killing capability.

TABLE II: Zero-Shot PUT 1 – Mutant Kill Summary

Total Mutants	Killed Mutants	Kill Rate (%)
16	13	81.25
17	15	88.24
14	13	92.86
16	15	93.75
16	16	100

TABLE III: Zero-Shot PUT 2 – Mutant Kill Summary

Total Mutants	Killed Mutants	Kill Rate (%)
10	10	100.00
10	7	70.00
10	8	80.00
10	8	80.00
10	9	90.00

TABLE IV: Few-Shot PUT 1 – Mutant Kill Summary

Total Mutants	Killed Mutants	Kill Rate (%)
14	14	100.00
14	11	78.57
16	16	100.00
15	15	100.00
15	15	100.00

TABLE V: Few-Shot PUT 2 – Mutant Kill Summary

Total Mutants	Killed Mutants	Kill Rate (%)
10	10	100.00
10	10	100.00
10	10	100.00
10	10	100.00
10	10	100.00

TABLE VI: Average Mutation Scores for Zero-shot and Few-shot Prompting Strategies

Prompting Strategy	Program P1	Program P2
Zero-shot	91.27%	84.00%
Few-shot	95.71%	100.00%

TABLE VII: Mutation score : Pynguin

Prompting Strategy	Program P1	Program P2
Zero-shot	87.5%	44.4%
Few-shot	87.5%	58.33%

IV. Data Analysis and Results

A. Data Validation

We verified that all test data was accurate and complete. Each method (Zero-shot, Few-shot, and Pynguin) was applied five times to both programs. The mutation score was calculated using the following formula:

$$\text{Mutation Score} = \left(\frac{\text{Mutants Killed}}{\text{Total Mutants}} \right) \times 100$$

B. Data Analysis

Few-shot prompting gave the best results in both programs, with 100% in PUT2 and 95.71% in PUT1. Zero-shot prompting was good in PUT1 (91.27%) but dropped in PUT2 (84.00%), showing it needs more guidance. Pynguin did okay in PUT1 (87.5%), but much lower in PUT2 (44.4%–58.33%), meaning it may not work well for all cases.

C. Outcomes (Results)

- RQ1: Can LLMs generate test cases that achieve competitive mutation scores compared to traditional tools like Pynguin?

- Yes, LLMs (specifically Gemini) generate test cases that achieve competitive mutation scores. In fact, Gemini based test cases, especially with the few-shot prompting method, outperformed Pynguin in both programs tested. The average mutation score for Gemini (few-shot) was significantly higher than Pynguin, demonstrating that LLMs can generate effective test cases.

- RQ2: Does few-shot prompting produce better test cases than zero-shot prompting?

- Yes, few-shot prompting produces better test cases than zero-shot prompting. The results showed that few-shot prompting achieved higher mutation scores across both programs. Specifically, for Program P1, the mutation score for few-shot (95.71%) was higher than zero-shot (91.27%), and for Program P2, few-shot achieved 100%,

while zero-shot dropped to 84%.

- RQ3: Can the refinement of LLM-generated test cases using MUTAP significantly improve their effectiveness?

- Yes, the refinement of LLM-generated test cases using MUTAP significantly improved their effectiveness. The MUTAP technique enhanced the test cases by targeting mutants that were not initially caught, leading to better mutation scores. The refined test suites using MUTAP performed better than the initial LLM-generated test cases, and the overall results indicated that this refinement process made the LLM-generated test cases more effective than traditional tools like Pynguin.

V. Conclusions

Few-shot prompting achieved the highest average mutation score across both programs, making it the most effective and consistent approach. It killed all mutants in PUT2 and nearly all in PUT1. Both few-shot and zero-shot prompting outperformed Pynguin, which showed comparatively lower scores—particularly in PUT2. Overall, few-shot prompting emerged as the best method, with LLM-based techniques clearly surpassing Pynguin in effectiveness.

References

[1] *arXiv preprint arXiv:2308.16557*, 2023, https://arxiv.org/abs/2308.16557.
[2] GitHub-Link: https://github.com/ExpertiseModel/MuTAP.