BEACHELOR THESIS

KOGNITIVE INFORMATIK

SYSTEMATIC GENERATION OF UNIT-TESTS WITH DIFFERENT LLMS AND PROMPT-DESIGN

RICHARD PAMIES

University of Bielefeld

SUPERVISED BY

PROF. DR.-ING. BENJAMIN PAASSEN,
M.SC. JESPER DANNAT

JULY 15, 2024

CONTENTS

1 INTRODUCTION

Unit-test are really important and useful. But most often those tests don't get written because of laziness, so the automatic generation of those tests arise as a possible solution.
Large Language Models (LLMs) offer for the the possibility of creating useful unit-tests. But those generated test are flawed because those LLMs dont work with reason, but instead the statistically most probable answer.
Prior work has tried under different circumstances to solve this problem.[paper 1,2,3].

My goal in this paper is to work out if prompt design by the LLM makes sense, and which LLMs, paired with which prompt design, provide the best answers.

2 BACKGROUND AND RELATED WORK

2.1 UNIT-TESTS

Unit-tests are tests written for already existing code to make sure the implemented functionalities still work after further work on the code is done.
The usefulness of unit-test is not controversial. Rather the time it takes to write them and the metrics used to test their usefulness.

Code coverage and statement coverage is often mentioned when talking about validating unit-tests. –Talking about the mechanisms of both and why they are not enough–

Here come other metrics i use in my work and their pros and cons:
mutation coverage
assert count and variaty

2.2 LLMS

An llm is only as good as its architecture and its training data. Generating unit-tests for mainstream code might be sufficient. But it is really easy to write code which covers niche subjects. And in those cases the LLM would need critical thinking, understanding and reasoning to fully understand the code and write useful and encompassing unit-tests.

2.2.1 *GPT*

2.3 RELATED WORK

3 METHODS

3.1 HIGH LEVEL DESCRIPTION OF MY APPROACH

I used the API-endpoints of different LLMs to send a structured prompt to those and receive code which should be the unit-test. Then i execute the unit
The sequence of my approach to generate these unit-tests is:

1. refining a prompt by the LLM
2. sending a prompt to the LLM
3. receiving solution and extracting the unit-test
4. executing the unit-test
5. re-sending the prompt with appended error msg if the unit-test fails
6. re-sending the unit-test with the instruction to remove the defective line

The structured prompt could look like this:

Lemma Let the coefficients of the polynomial

$$a_0 + a_1 x + a_2 x^2 + \cdots + a_{m-1} x^{m-1} + x^m$$

be integers. Then any real root of the polynomial is either integral or irrational.

This high level description what it should accomplish detailed description for reproduction for experienced user use illustrating diagrams as much as possible (no source code, but perhaps pseudo code) provide a good reason for each design choice(source or math) use a running example to illustrate the steps

3.2 PROVIDE OVERVIEW DIAGRAM ILLUSTRATING MY STEPS

4 IMPLEMENTATION

4.1 FOR EACH OF THE STEPS HOW IT WAS IMPLEMENTED IN THE CODE

5 EXPERIMENTS

5.1 WHAT I WANT TO SHOW

5.2 HOW I WANT TO SHOW IT

5.3 BASELINES AND ASSUMPTIONS

5.4 WHAT WAS MY INITIAL ASSUMPTION

5.5 TABLE FOR EXPERIMENTS

5.6 DISCUSSION OF THE RESULTS

6 CONCLUSION

7 LATEX FEATURES

sparse[1]

Did you know? The discrete fourier transformation shown in equation 7.1 is the backbone of the modern information-society.

$$f_m \tag{7.1}$$

For words of science, see [**botsch2010polygon**]. Unfortunately, that book has nothing about wolves.

1 This is a footnote