Automated Prompt Generation for REST API Test Amplification Using LLMs

Scope description

January 14th 2025

Giorgi Guledani
University of Antwerp
Computer Science Department
Antwerp, Belgium
giorgi.guledani@student.uantwerpen.be

***Abstract*—With the recent release of Restats, a semi-automated coverage tool for REST APIs, evaluating test coverage from a black-box approach has become easier. The thesis aims to leverage this tool to train a model that associates test cases with the most effective prompts, using large language models like ChatGPT. The main goal is to retrieve the best prompts to amplify tests from given test suites and OpenAPI documentation, leading to improved RESTAPI coverage. Additionally, the model will offer valuable insights into prompt engineering, helping refine the prompts for better amplification and coverage output.**

1. Introduction

Writing tests for a REST APIs can be a very time-consuming task for developers [1]. Recently, Lopez et al. [2] introduced standardized REST API coverage criteria for black-box testing. These criteria laid the foundation for the development of the Restats tool [3]. The tool allows for quantitatively measuring the coverage of REST API test cases using a black-box approach. This brings the idea that Restats can also be used as part of evaluating a model during training to associate the best prompt for a given test case. If the coverage increases after test amplification, it would be assumed that an existing test suite has improved with the added test(s) and that a specific prompt proved to be a success in amplification.

Another recent study, also by Lopez et al. [4], compared a white-box approach (EvoMaster [5]) with a black-box approach (RestTestGen [6]) for generating tests. They observed that a combination of black-box and white-box yielded the best results in terms of code coverage and fault finding. A combination of both approaches can be realized in this thesis by including both OpenAPI documentation and/or source code in the prompts.

Additionally, tools like ChatGPT provide the option to adjust the randomness temperature of generated outputs. This can be experimented on in two ways:

1) Keeping randomness would increase the diversity of generated test cases, allowing for greater exploration with unsuccessfully generated tests.
2) Eliminating randomness would increase predictability, allowing for greater understanding of the effectiveness of specific prompts.

2. Relevance

There are multiple ways to generate REST API test cases, which means generating tests from scratch. One such tool is RestTestGen [6], which generates test cases for REST APIs from OpenAPI specifications using a black-box approach. EvoMaster [5] takes it a step further by using a white-box approach, and generates test cases using specific algorithms. Because of the white-box approach, EvoMaster evaluates the code using code coverage (statement and branch coverage), independent from the coverage evaluation proposed by Lopez et al. [2]. A downside of this tool is that it requires more manual work, in the sense that the user has to write their own class that extends the one in their library.

There are no tools available for amplifying REST APIs: generating tests from existing ones. Previous state-of-the-art tools generate tests from scratch. It has been discussed [1] that adding new tests based on existing ones, can make the test generation process more targeted and cost-effective. On the one hand, the test generation process can be geared to-

wards achieving a specific engineering goal better based on how existing tests perform with respect to the goal.

From previous research utilizing LLMs for test amplification, using 3 different prompts has already shown to be successful for amplifying tests [7]. Training a model that can associate the best prompt from a large set of prompts for a given code snippet could prove to be very powerful. Another benefit is that it could reveal valuable insights for prompt engineering for future research.

3. Research Question

The research question that we will focus on during the thesis is one of the following:

- *How can a prompt generation model be trained to amplify REST API tests using LLMs, where they increase REST API coverage from a black-box approach?*
- *In what way can a prompt generation model trained with LLMs enhance REST API test amplification and increase black-box test coverage?*
- *How can LLM-leveraged prompt generation models be trained to optimize prompt generation for improving REST API test coverage in black-box testing?*
- *How can prompt generation models be trained for improving REST API test coverage?*

4. My qualifications for the thesis

I specialize in Data Science and AI, making my background well-suited to the requirements of the thesis. In addition, I have taken a course in Software Engineering and Software Reengineering, which has provided me with a deep understanding of the importance of tests in software development and maintenance. Through the reengineering course, I learned various software design patterns and techniques [8] that will be useful in reengineering the Restats tool.

Furthermore, I have completed a course in Reinforcement Learning. This could prove to be useful in designing and training the model that can generate effective prompts, and improve REST API test coverage through feedback loops.

5. Challenges and approaches

5.1. Semi-automated REST API coverage tool

The Restats tool at present is only semi-automated and relies on executed test logs (requests and responses) and OpenAPI documentation of the targeted application:

- We use generated test code for interacting with the API. These interactions need to happen through a proxy. To get coverage results for a test suite, all test cases must interact with the REST API first, which you must manually set up from the tool. To automate this process, all tests should be executed and routed through the proxy.
- **Burp Suite**: Used as a proxy between the testing framework and REST API. These interactions are logged and bumped to HTTP requests and responses in text format, which is required as input for the Restats tool. To automate this part, the proxy should be in the source code itself.

Before training of the model can even be considered, the tool will need to be reengineered. The biggest time-consumer is generating HTTP dumps through Burp Suite, which we will automate.

5.2. Initial prompts for training

Training the model would need initial prompts to pick the most suited one for a given test case. These will have to be manually written.

Manually writing prompts thousands of prompts would take a lot of time. Performing data augmentation is ideal, as we can greatly increase the initial, manually written prompts. We came up with the idea to use 2 sets of prompts, and have each prompt from each set form a pair with the other set. This means that given 2 sets of prompts A and B, we'd produce:

$$\#prompts = |A| \times |B|$$

Given a mere 10 prompts in each set, this would mean that 100 prompts of initial training data will be augmented.

For example:

- Prompt 1 in set A:
  ”Play the role of an experienced software tester for REST APIs and perform test amplification.”
- Prompt 2 in set B:
  ”Here is the test case: **test case** and here is the full OpenAPI specification: **specification**.”

Combining various prompts could give interesting insights into when specific prompts perform the best and why.

5.3. Test amplification scenarios

We identified 4 test amplification scenarios:

- *Easy*: No additional context is needed; the test case alone is enough for amplification.
- *Medium*: Amplification requires context from functions or variables within the same file.
- *Hard*: Amplification requires context from functions or variables spread across different files.

- *Very Hard*: Amplification requires deep context from external libraries or APIs, along with complex interactions between multiple systems or asynchronous processes.

Amplification in *Hard* scenarios may be inconsistent due to the input limits of LLMs. If the test case requires extensive context across different files, it might not be possible to provide all the necessary information at once. Alternative approaches could involve breaking down the context into smaller, more manageable chunks for feeding information. Amplification for *Very Hard* scenarios is considered to be unfeasible.

6. Planning

Implementation and evaluation of the model will occur incrementally. We will follow the plan as shown below. The plan is subject to change during the year:

Phase		Period
1)	Literature study	October - December 14th
2)	Restats tool reengineering	December 14th - January 7th
3)	Training data collection	January 7th - January 21th
4)	Implementation scenarios Easy & Medium	January 21th - March 14th
5)	Evaluation scenarios Easy & Medium	March 14th - March 28th
6)	Thesis + Implementation scenarios Hard	March 28th - April 14th
7)	Thesis + Evaluation (full)	April 14th - April 28th
8)	Thesis	April 28th - June 16th

Figure 1. Thesis plan

In the first phase, a literature study is done to get a better view of what was already tried, and what can be improved on.
The second phase will be about changing up the Restats tool to fit our needs. The current tool lacks automation, so it will require reengineering.
The third phase will be about finding suitable projects to train the model on, and prompt engineering to obtain initial prompts.
The fourth phase will involve implementing the model to amplify tests in *Easy* and *Medium* test case scenarios.
In the fifth phase, the model will be evaluated by comparing it to popular tools such as EvoMaster and RestTestGen. Selected projects will be used as benchmarks, and the resulting test coverage will be analyzed and compared across these tools. Another interesting comparison could be to apply the model to the same projects used by Bardakci et al. [7] and observe whether the model managed to give the same or even better prompts.
Starting from the sixth phase, thesis writing will begin as we will already have usable results. Concurrently, we will implement the model to handle *Hard* test case scenarios.
In the seventh phase, the previous evaluation will be extended to include the *Hard* test case scenarios.
In the final phase, we will focus entirely on completing the thesis.

References

[1] Benjamin Danglot, Oscar Vera-Perez, Zhongxing Yu, Martin Monperrus, and Benoit Baudry. The emerging field of test amplification: A survey. *CoRR*, abs/1705.10692, 2017.

[2] Alberto Martin-Lopez, Sergio Segura, and Antonio Ruiz-Cortés. Test coverage criteria for restful web apis. In *Proceedings of the 10th ACM SIGSOFT International Workshop on Automating TEST Case Design, Selection, and Evaluation*, A-TEST 2019, page 15–21, New York, NY, USA, 2019. Association for Computing Machinery.

[3] Davide Corradini, Amedeo Zampieri, Michele Pasqua, and Mariano Ceccato. Restats: A test coverage tool for restful apis. In *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 594–598, 2021.

[4] Alberto Martin-Lopez, Andrea Arcuri, Sergio Segura, and Antonio Ruiz-Cortés. Black-box and white-box test case generation for restful apis: Enemies or allies? In *2021 IEEE 32nd International Symposium on Software Reliability Engineering (ISSRE)*, pages 231–241, 2021.

[5] Andrea Arcuri. Evomaster: Evolutionary multi-context automated system test generation. In *2018 IEEE 11th International Conference on Software Testing, Verification and Validation (ICST)*. IEEE, April 2018.

[6] Emanuele Viglianisi, Michael Dallago, and Mariano Ceccato. Resttestgen: Automated black-box testing of restful apis. In *2020 IEEE 13th International Conference on Software Testing, Validation and Verification (ICST)*, pages 142–152, 2020.

[7] Tolgahan Bardakci, Serge Demeyer, and Mutlu Beyazıt. Test amplification for rest apis using large language models.

[8] Serge Demeyer, Stephane Ducasse, and O Nierstrasz. *Object-Oriented Reengineering Patterns*. The Morgan Kaufmann Series in Software Engineering and Programming. Morgan Kaufmann, Oxford, England, July 2002.