COMP0103 - CW2

David Roman Frischer

April 7, 2024

Contents

1 Introduction

This project explores the potential of large language models (LLMs) to automatically generate solutions to LeetCode interview problems in Java. But the twist? We're focusing on smaller, more portable models, and investigating whether they can compete with their larger competitors while running efficiently on a personal computer.

- **Model Efficiency:** Can LLMs effectively produce correct and functional Java code for LeetCode problems?
- **Quick Model Improvement:** Can smaller models achieve better results using an iterative and bounded refinement strategy inspired from "Self-Refine" [1] ?

To address these questions, we'll create a pipeline that tackles equally all three subproblems outlined in the coursework description: prompting construction, extracting the relevant code snippet, and verifying its functionality.

2 Technical Approach

2.1 Data Source

The project will use a public repository LeetCode database question made by interviewcoder on GitHub[1]. This repository gathers over 200 Java interview questions with matching JUnit tests. Its structure, with separate folders for code and tests, makes it good for automating the pipeline.

2.2 Models

We will evaluate multiple large language models (LLMs) to compare the quality of their generated code snippets and their overall correctness. Ollama[2] software will be used to efficiently pull and run these models. Table 1 details the potential models chosen for this research.

Model	Size (GB)	Parameters (B)	Quantisation
gemma:7b-instruct	5.2	9	4-bit
llama:7b	3.8	7	4-bit
mistral:7b-instruct	4.1	7	4-bit

Table 1: List of models to be compare

2.3 Technology use

This project use several key technologies:

- **Ollama:** This software facilitate the process of running LLMs locally. It pulls pre-trained models from its library[3] and serves an API for querying them on a local computer.
- **Python:** While the generated LeetCode solutions will be written in Java, the pipeline itself will be developed in Python. This choice ease the development and rapid prototyping.
- **Infer from Facebook (Meta):** As recommended in the coursework, Infer will be used for static analysis of the generated code. It helps identify potential compilation errors but cannot guarantee the code's semantic correctness (meaning it might compile but not pass the unit tests).

[1] `https://github.com/interviewcoder/leetcode`
[2] `https://ollama.com/`
[3] `https://ollama.com/library`

2.4 Pipeline

Figure 1 illustrates the pipeline's workflow for generating, extracting, verifying, and testing solutions to LeetCode problems.

1. **Generate the solution:** The pipeline starts with a Java file named "Practice.java" containing the problem description and an empty class to be implemented. Using Python, an API request is sent to the LLM, prompting it to solve the problem and generate the Java code.
2. **Extract and Insert Code:** The LLM's response is parsed to extract the relevant Java code snippet. This snippet is then inserted into the "Practice.java" file.
3. **Compile and Verify:** The updated "Practice.java" file is compiled and verified using Infer from Facebook. If the compilation fails, the pipeline goes to step 4, otherwise it continues to step 5.
4. **Feedback:** If the number of retries is less than the value of `MAX_RETRIES`, the LLM integrates the error message into its context so that it can refine its response back to step 1. Otherwise it jumps to step 6 to generate the report.
5. **Test the Solution:** The unit tests provided in the repository are run against the generated code to assess its validity. If a test fails, the process jumps back to step 4 to refine its response.
6. **Generate Report:** The pipeline generates a final report that captures the entire process, potentially containing multiple iterations if steps 3 or 4 failed in previous attempts. Each iteration within the report will include:
   - The prompt sent to the LLM
   - The generated code of the LLM
   - The extracted code snippet from the LLM's response
   - The updated "Practice.java" file
   - The compilation results
   - The path to the verification folder of Infer
   - The feedback response of the LLM
   - The test results (potentially empty if compilation failed)

3 Challenges and Risks

The project will face several challenges to highlight its complexity

1. **Time Constraints:** Building a large pipeline in a limited timeframe. To mitigate this risk, I may ask a colleague to join me in this work.
2. **Prompt Engineering:** Making effective prompts for the LLM to understand the LeetCode problems and generate accurate solutions can be difficult. Fine-tuning the prompts might be necessary to achieve better results.
3. **Code Insertion Accuracy:** While extracting the code snippet might be a bit challenging, the difficulty lies in creating a valid Java file by correctly inserting the extracted code.
4. **Small Model Capabilities:** Smaller LLMs might underperform compared to their larger counterparts. We anticipate that the generated code might have compilation errors or produce an incorrect solution. To the possibility of not even passing the compilation step. We could use GPT3.5 to at least confirm that the pipeline is feasible and to show that smaller models are not good enough to complete it.
5. **Data Collection and Analysis:** While a key objective is to generate a significant number of reports, it's crucial to ensure the data collected is meaningful to raise interesting conclusion from this work.

References

[1] Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023.

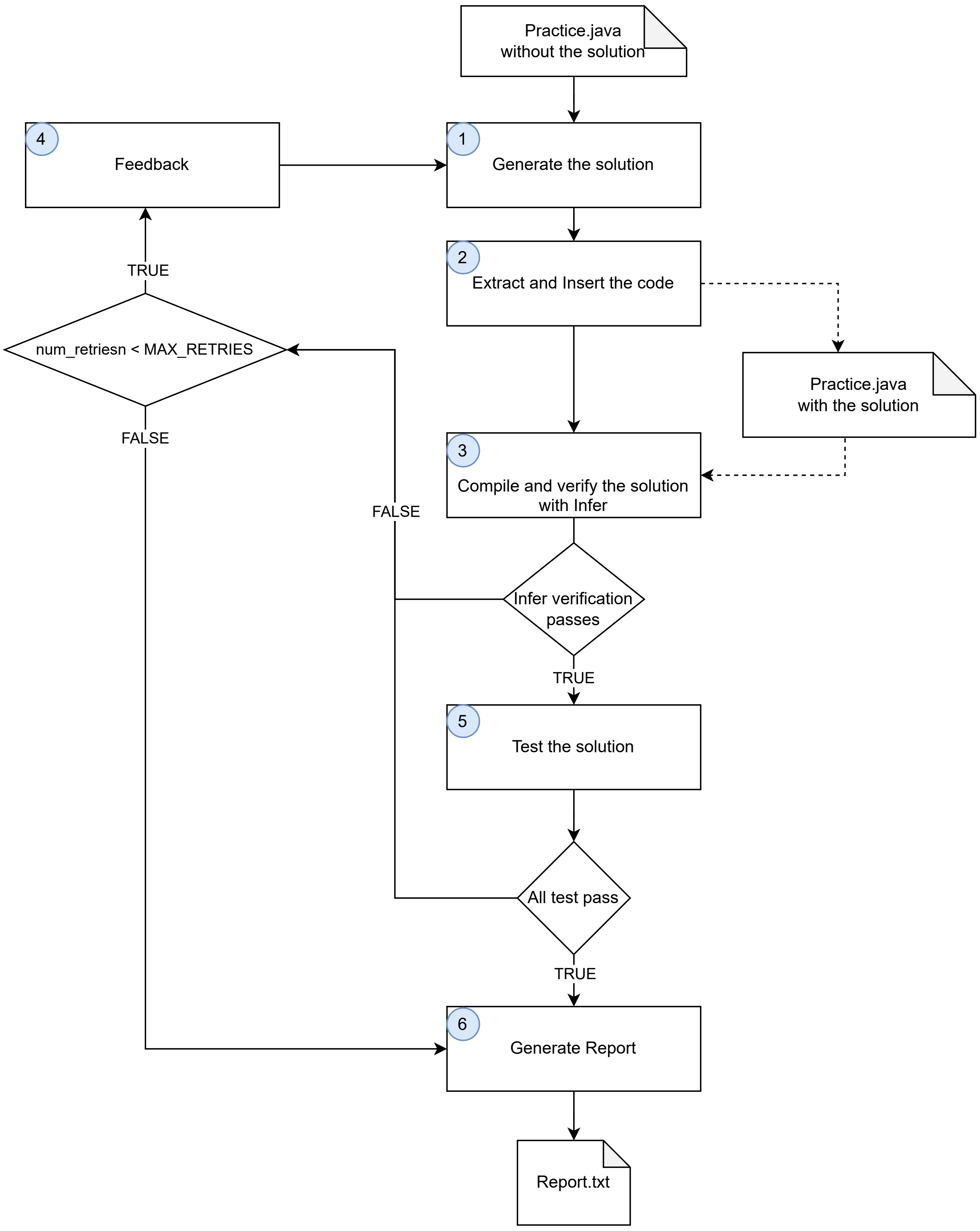
Practice.java without the solution
1
Generate the solution
4
Feedback
2
Extract and Insert the code
TRUE
num_retriesn < MAX_RETRIES
Practice.java with the solution
FALSE
3
Compile and verify the solution with Infer
FALSE
Infer verification passes
TRUE
5
Test the solution
All test pass
TRUE
6
Generate Report
Report.txt

Figure 1: The pipeline LeetCode solver