LLM-based Software Engineering Agent tested on SWE-bench

Intro to Large Language Models (BITS F471)
Second Semester 2024-25

Course Project
by

Repopotamus
Neha Gujjari (2023A7PS0011P)
Varun Vijay (2023A7PS0016P)
Nishant Pradhan (2023A7PS0030P)
Amrit Lahari (2023A4PS0442P)

May 2025

Contents

Abstract

We built an LLM-based agentic workflow to generate code patches for reported bugs in various public Github repositories. These bugs were taken from the SWE-bench Lite dataset and include well-known repositories like django, scikit-learn, matplotlib, etc. The patches generated by our model were unit tested by the raw, inbuilt SWE-bench evaluator. The results of this evaluation give details on how many test cases each patch passed.

1. Introduction

SWE-bench is the biggest and only major metric available for software engineering agents. It focuses on real-world problems and tests AI models' ability to solve actual Github issues. The bugs that need to be fixed are compiled into multiple datasets on SWE-bench, Lite, Multimodal, Verified, and Multilingual. We chose Lite to test our model on because it is computationally less expensive. It provides a smaller subset of 300 tasks from the full benchmark and aims at maintaining difficulty while focusing on more functional bug fixes.

2. Dataset: SWE-bench

The tasks in the SWE-bench dataset are in the form of instances. Each instance contains:

- The GitHub repository name
- A unique instance ID
- Base commit version
- Problem statement
- Human-verified ground truth patch
- Hints to solve the issue
- FAIL_TO_PASS (bugs which need to be addressed)
- PASS_TO_PASS (interlinked functionalities which must not be broken)
- Test patch (create an environment to test the bug, not used by the model)

The dataset is divided into two parts: dev and test. Dev is a validation set which could be used for hyperparameter tuning. The test set contains the actual test cases the model is evaluated on.

3. Methodology

3.1. Preprocessing

The dataset was imported using the hugging-face dataloaders in Python. Only the test split of the dataset was accessed. The test_patch and hints_text columns were dropped from the data set to reduce the context given to the model. The necessary file was retrieved using the repository name and patch fields. The base commit version was also used to restore the correct version of the file using the "git checkout" command from code.
The patch field contains the ground truth answer, so this was not passed to the model. It was only used to extract the relevant file path. This was done to reduce token usage by cutting down LLM calls. The necessary fields from the instance and the code from the Python file were passed to the model for patch generation.

3.2. Initial Patch Generation

A detailed verbose system prompt was invoked by the model containing all the necessary instructions. In this step, the model only had access to the problem statement, original code and required test cases.

3.3. Reflection

An LLM call takes a given patch as an input and checks it against the problem statement and the test cases. This reflection was recursively called on each patch three times to ensure maximum accuracy with minimum computation.

3.4. Context Matching

This call was dedicated to checking that the correct line numbers are being fixed and making changes if needed. This was done to ensure that the patch does not edit non-existent lines. This was also called recursively three times.

3.5. Patch Refining

This final call was intended to ensure that the final patch was in a proper unified diff format and that there were no unnecessary newlines and white spaces.

3.6. Final Model Output

The result of each instance was given as a JSON object containing the instance ID, repository name and generated patch. All of these results were written to a JSON file separated by newlines for evaluation.

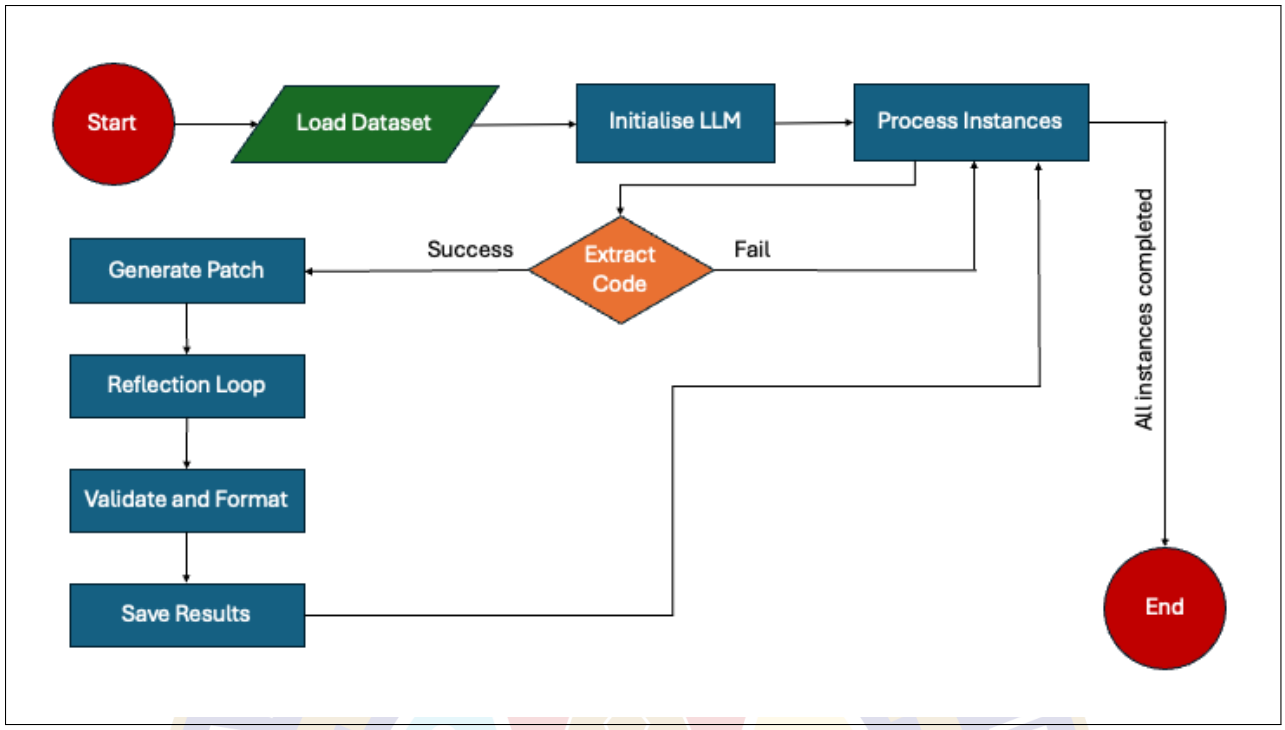

Figure 1: Workflow

4. Evaluation

The patches were individually evaluated using the built-in "run_evaluation.py" script file from SWE-bench. Each instance was run on a separate docker container in parallel.
The evaluation uses the process of "git apply" to apply the patches before checking for logical correctness. Application checks that the patch matches the surrounding code in the file by matching it to the ground truth.
On successful application, a report is generated for each instance detailing all the FAIL_TO_PASS and PASS_TO_PASS conditions that the patch succeeded at.

5. Results

Our model achieved 11.67% accuracy on all 300 instances in the dataset. This means that for 35 of 300 instances, the patch applied and all required test cases passed.
In 144 of 300 instances, there were patch application errors, mostly due to formatting and context issues. 156 of 300 of the patches generated by our model applied correctly, but 121 of these patches were unresolved, meaning the patch did not pass all the necessary test cases.
It should also be noted that there were zero successful instances for the repositories: flask, pylint and sphinx. These were the only repositories for which the cloning links seemed unreliable. This could be the cause for error in these cases.

No. of instances	Unapplied	Applied	Unresolved	Resolved
300	144	156	121	35

Figure 2: Overview of Results

6. Future Work

We were severely constrained by the token usage limit. The following are some changes we feel might improve our model:

- Using stronger models
- Increasing the number of reflection loops
- Adding a cosine temperature scheduler to change the temperature of the model for each iteration
- Creating a multi-agent workflow to simulate an actual workspace in terms of review and feedback
- Automated file extraction through LLM calls rather than string formatting
- Adding web searching capabilities in case of successive failures
- Adding a frontend to test on a repository of the user's choice

7. Conclusion

This project explored using an LLM to generate patches for SWE-bench Lite bug reports. A multi-step agentic approach enabled context-aware and logically correct fixes. The model successfully produced and valid patches as evaluated by SWE-bench. Results show that with continued refinements, LLMs can contribute meaningfully to automated bug fixing across diverse real-world software projects.