Gen AI AI-Powered Code Refactoring & Dependency Updater

Project Overview

Maintaining large codebases is challenging due to **outdated dependencies, code quality issues, and evolving best practices**. This project explores how **Generative AI can automate code refactoring, dependency management, and testing**, making software maintenance more efficient.

The goal is to build an **AI-powered software engineering agent** that:

1. **Clones an existing GitHub repository** and processes it for analysis.
2. **Reviews code quality**, suggesting improvements based on best practices.
3. **Refactors and modernizes the code**, ensuring consistency across functions and files.
4. **Updates dependencies** by reading `pyproject.toml` or `requirements.txt`, identifying outdated packages, and upgrading them to their latest stable versions.
5. **Ensures compatibility** by testing the refactored code and verifying dependency upgrades.
6. **Uses multi-agent collaboration** to divide tasks efficiently (e.g., code review, refactoring, testing).

By leveraging **LLMs, code analysis tools, and multi-agent collaboration**, this project automates the process of improving a software repository with minimal human intervention.

Why Synthetic Data?

To evaluate the **robustness and correctness** of the AI agent, we will create **synthetic repositories** with:

* **Intentional code smells** (e.g., duplicate code, unused variables, poor error handling).
* **Outdated dependencies** to test automatic upgrades.
* **Messy project structures** to test file organization improvements.

Note that the following is simply an example of poor code to unit test your solution, once you have a solution you need to showcase it working on [AWS FMBench Orchestrator](https://github.com/awslabs/fmbench-orchestrator) repo. You can fork this repo, have your agent work on it and submit a PR once you have tested the new and improved code written by your agent.

Example synthetic repo files:

1\. Python Code with Outdated Practices

`app.py`

python
CopyEdit
`import requests  # Older version, missing timeout`
`import json`

`def fetch_data(url):`
    `response = requests.get(url)  # No error handling`
    `return json.loads(response.text)`

`print(fetch_data("https://api.example.com/data"))`

Issues:

* No timeout in `requests.get()`.
* No exception handling.
* Outdated `requests` version.

2\. Old Dependency File

`requirements.txt`

txt
CopyEdit
`requests==2.22.0`
`numpy==1.18.5`
`flask==1.1.1`

Task:

* Find the latest versions and update them.

3\. Complex Codebase with Poor Structure

Multiple files with inconsistent coding styles, missing docstrings, and unused functions.

Technical Components & Implementation

1. **Multi-Agent System for Code Analysis & Refactoring**

* **Agent 1 (Code Review Agent):** Analyzes code quality, suggests improvements.
* **Agent 2 (Refactoring Agent):** Rewrites functions to improve maintainability.
* **Agent 3 (Dependency Updater):** Reads dependency files, finds updates, and modifies them.
* **Agent 4 (Testing Agent):** Runs unit tests to ensure compatibility after changes.

2. **Code Analysis & Modernization**
   * Use **LLMs (e.g., GPT-4, Code Llama, DeepSeekCoder)** for **automatic code improvement**.
   * Refactor for **better readability, efficiency, and security**.
   * Maintain **functionality across refactored files**.
3. **Dependency Management & Upgrades**
   * Read `requirements.txt` or `pyproject.toml`.
   * Query the latest versions using **PyPI APIs**.
   * Upgrade dependencies, check for **breaking changes**, and adjust code accordingly.
4. **Testing & Validation**
   * Run **unit tests before and after** code changes.
   * Generate synthetic test cases for new functions.
   * Verify that all upgraded dependencies work correctly.

Evaluation & Success Metrics

* **Reduction in technical debt** (e.g., code complexity scores).
* **Correctness of dependency upgrades** (e.g., successful package builds).
* **Improved maintainability** (e.g., adherence to PEP 8, linting scores).
* **Successful test execution** post-refactoring.

Why This Project Matters?

This project showcases how **GenAI can assist software maintenance by automating code reviews, dependency management, and refactoring**. It applies **multi-agent AI collaboration** in a practical software engineering scenario.

Code Repository for Real-World Testing:
🔗 [AWS FMBench Orchestrator](https://github.com/awslabs/fmbench-orchestrator)