Lecture 5: Building A CLI Agent

Mia Mohammad Imran

Overview

- Feedback on Paper Review and Hands on Activities
- How to build a CLI Agent
- SWE Agent

Week 2: Paper Review Feedback

- Do not use ChatGPT (or similar tools) to generate summaries.
  - Write in your own words.
  - **Minor mistakes are okay.**
  - **AI-generated patterns are easy to spot:** many submissions with same points, wording, mistakes, or cited papers. *Given the depth of the long papers, I would expect more variety than the ones language models pick*
  - Please don't make the assessment harder than it needs to be
- Since most of you have talked about LLM non-deterministic as limitations, you can check this guideline on how researchers are approaching to reduce this problem: https://llm-guidelines.org/

Week 2: Hands-on Activity Feedback

- The idea was to make prompts more robust and generic
- When you update a prompt, think about the patterns you observe and not one specific problem
- Do not mention a specific problem on your prompt. That will harm generalization
- Think about the unseen cases that may arise in future and not only the validation set you have right now

What Is a CLI Agent

A CLI agent is a program that:

- **Accepts** user commands from the terminal
- **Interacts** with an LLM through structured prompts
- **Performs** actions on the file system or repository
- **Iterates** until a goal condition is met or fails safely

The Core Loop (ReAct Pattern)

The agent operates on a continuous cycle:

- **Observe:** Read files, list directories, or view compiler errors
- **Think:** Analyze the observations against the goal
- **Act:** Execute a command (e.g., sed, python, git)
- **Verify:** Check if the action moved toward the goal

High-Level Architecture

A. The CLI Interface (The Entry Point): Handles the user's intent

- **Command Parsing:** Arguments like --model, --repo-path, and the specific task description
- **Session Management:** Maintaining a history of what the agent has done so far

B. The Prompt Controller (The Brain): This is not a single prompt, but a **stateful template**

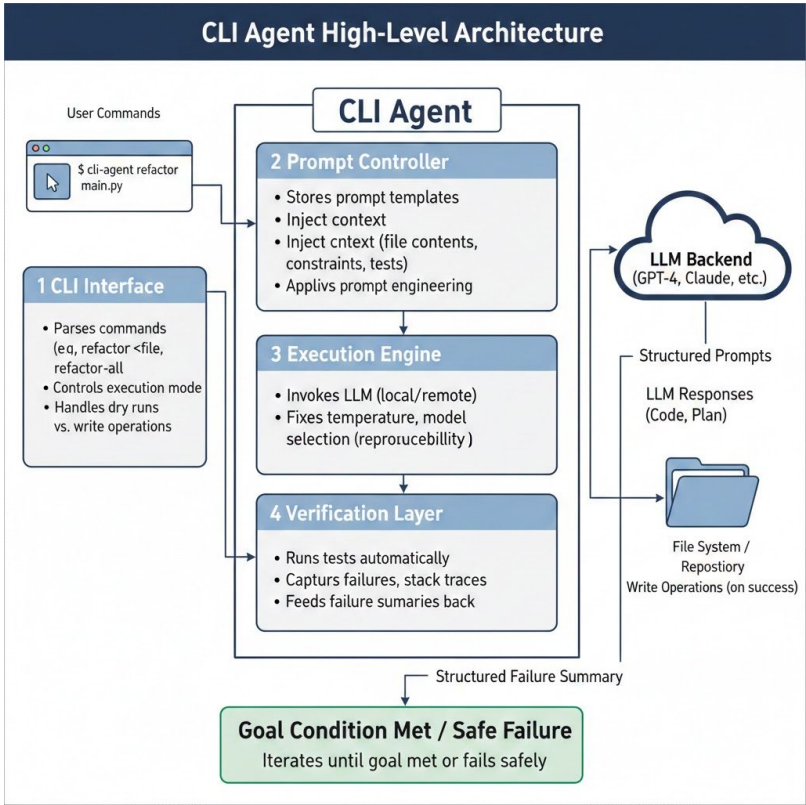
- **System Prompt:** Defines the agent's identity ("You are a world-class Software Engineer") and the available tools
- **Context Injection:** Dynamically pulls in the current working directory, file structure, and relevant code snippets

C. The Execution Engine (The Hands): This layer translates LLM "thought" into "action"

- **Toolbox:** A set of predefined functions the LLM can call (e.g., search_codebase, edit_file, run_tests)
- **Sandboxing:** Good idea is to run inside **Docker container**. This prevents the agent from accidentally deleting your host files or executing malicious code

D. The Verification Layer (The Safety Net): An agent is only as good as its ability to admit failure

- **Test Integration:** Automatically running pytest or npm test after a change
- **Linter Feedback:** If the LLM generates a syntax error, the Verification Layer captures the stderr and feeds it back to the agent for "self-healing"



Agent Control Loop

The core idea of agentic behavior is a **loop with memory and feedback**:

- **Observe:** Read code, tests, and constraints.
- **Decide:** Generate a refactoring proposal.
- **Act:** Write updated code to disk.
- **Evaluate:** Run tests and collect results.
- **Reflect:** Decide whether to retry, stop, or escalate.

Prompt Design for CLI Agents

Key differences from chat prompts:

- Prompts must be **machine-oriented**, not conversational
- Output format must be strict
- Instructions must be invariant across runs

Example constraints:

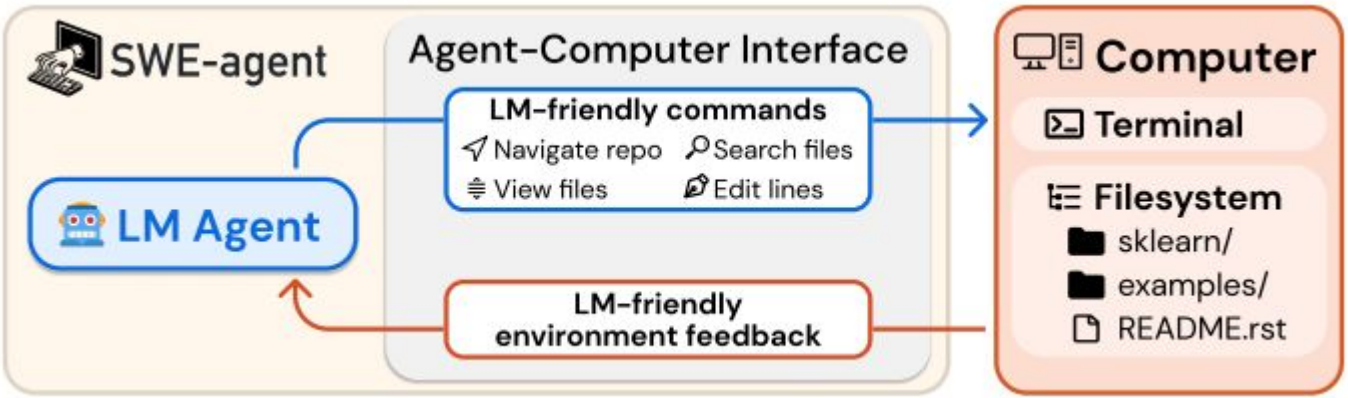
- Preserve function signatures
- Do not introduce new dependencies
- Do not modify tests
- Output code only, no explanations

Challenges in Engineering CLI Agents

- **Context Window Limits:** Large repositories exceed LLM limits; agents must learn to "search" rather than "read everything"
  - RAG can be helpful to feed selective history than complete context
- **Infinite Loops:** Agents might get stuck trying the same failing fix.
  - Implementation of a **Max Iteration** cap/setting a **Threshold** can be helpful
- **Cost:** Autonomous loops can consume tokens rapidly
  - Selection of model matters for cost/GPU

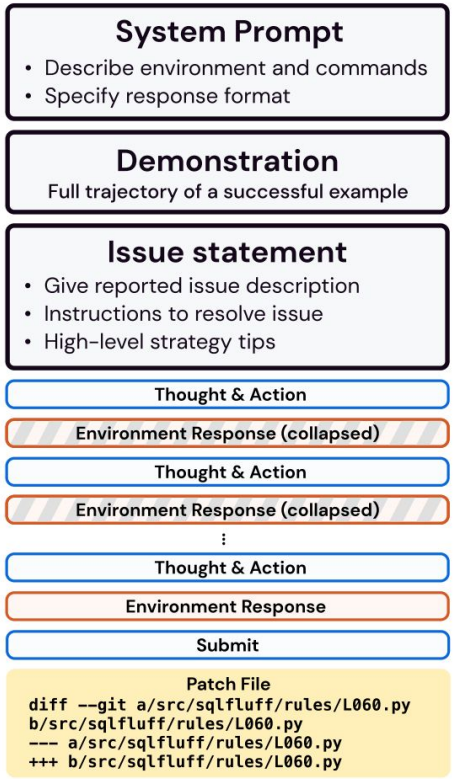
Case Study: SWE-Agent

- The **SWE-agent** (Software Engineering Agent) is a good example of this architecture
- It is designed to turn GitHub issues into pull requests automatically
- **Key Components in SWE-Agent:**
  - **ACIs (Agent-Computer Interfaces):** Instead of giving the LLM a raw terminal, SWE-agent provides simplified tools. For example, instead of using vim, it uses a specialized edit command that handles line-based replacements, making it harder for the LLM to "hallucinate" file content
  - **Stateful Browsing:** It maintains a "window" of the file, allowing it to scroll up or down



Case Study: SWE-Agent

- https://swe-agent.com/latest/
- https://proceedings.neurips.cc/paper_files/paper/2024/file/5a7c947568c1b1328ccc5230172e1e7c-Paper-Conference.pdf
  - https://github.com/SWE-agent/mini-SWE-agent
  - https://github.com/SWE-agent/SWE-agent



Demo

https://github.com/imranraad07/CS-5001-AI-Augmented-SE/tree/master/Week_3/demo/cli_agent

- A simple agent that has 4 commands (with lots of bugs!)
  - Generate codes
  - Generates tests
  - Report tests
  - Commit the changes
- It’s not exactly a “Complete” AI Agent yet
  - It does not do **Reflect**
  - It does not save history
  - It’s not Chat format, rather command line i.e., each CLI command is different