Agent Workflow

Table of contents

1 Agent Workflow

This document explains the internal workflow of the `code-agent` and how it processes your requests. Understanding this flow can help you interact with the agent more effectively and troubleshoot issues.

1.1 The Agent's Core Loop

The `code-agent` operates as an interactive loop, driven by a LangGraph `StateGraph`. This graph orchestrates the interaction between the user, the Language Model (LLM), and the agent's tools.

Here's a simplified overview of the workflow for each user query:

1. User Input: You provide a natural language query to the agent.
2. Context Retrieval: The agent first performs a similarity search against its persistent conversation history (stored in `ChromaDB`). Relevant past interactions are retrieved and added to the current context.
3. LLM Invocation (Reasoning): The LLM receives the user's query along with the retrieved context and a set of tool descriptions. Its task is to decide the next best action:

- **Use a Tool**: If the LLM determines that a tool is needed to fulfill the request (e.g., creating a file, reading code, generating tests), it outputs a `ToolCall` specifying the tool's name and arguments.
- **Direct Response**: If the LLM can answer the query directly without needing a tool, it generates a natural language response.

4. **Tool Execution**: If a `ToolCall` is made, the specified tool is executed. The tool performs its action (e.g., creates a file, modifies code, runs a script) and returns its output.
5. **LLM Re-invocation (Tool Output Processing)**: The LLM receives the output from the executed tool. It then uses this new information to decide the next step:
   - **Further Tool Use**: If the goal is not yet achieved, the LLM might call another tool or the same tool with different arguments.
   - **Final Response**: Once the LLM determines the user's request has been fulfilled, it generates a final natural language response.
6. **History Update**: Both the user's query and the agent's final response (including any tool outputs) are stored in the persistent `ChromaDB` vector store, enriching the agent's memory for future interactions.

This loop continues until the agent provides a final response to the user.

1.2 System Prompt and Agent Persona

The agent's behavior and decision-making are heavily influenced by its **system prompt**. This prompt defines the agent's persona, its capabilities, and its instructions for interacting with the user and using its tools.

The `code-agent` is instructed to act as a proactive and autonomous "senior software engineer." It is encouraged to use its tools directly to achieve user goals without constantly asking for explicit instructions or claiming it cannot perform a task if a tool is available.

1.3 Persistence and Memory

The `code-agent` maintains conversational memory across sessions using a local `ChromaDB` vector store.

- **Location**: The memory is stored in a `.code_agent_memory` directory within your project's root.
- **Mechanism**: Each user query and agent response is embedded and added to this vector store. When a new query arrives, the agent retrieves semantically similar past interactions to provide relevant context to the LLM.
- **Benefts**: This allows the agent to remember past tasks, learn from previous interactions, and maintain a more coherent conversation flow over time.

1.4 Example Interaction Flow

Consider the user query: "Create a new Python file named `hello.py` and add a function `greet()` that prints 'Hello, World!'."

1. **User Input**: "Create a new Python file named `hello.py` and add a function `greet()` that prints 'Hello, World!'."
2. **LLM Reasoning**: The LLM analyzes the request and determines that the `new-file` tool is appropriate. It formulates the `file_path` and `content` arguments.
3. **Tool Execution**: The `new-file` tool is called with `file_path='hello.py'` and `content='def greet():\n    print("Hello, World!")'`.
4. **Tool Output**: The tool successfully creates the file and returns a message like " Successfully created hello.py".
5. **LLM Final Response**: The LLM receives the tool's output, confirms the task is complete, and generates a final response to the user: "I have created `hello.py` with the `greet()` function."
6. **History Update**: The entire interaction (user query, tool call, tool output, agent response) is saved to `ChromaDB`.

This iterative process allows the agent to break down complex requests into manageable steps, execute them, and provide a comprehensive response.