Project_P1: Towards Intelligent Assistance: A Natural Language Terminal

New Attempt

- Due Oct 3, 2025 by 11:59pm
- Points 100
- Submitting a file upload
- File Types zip

Learning Objective

This is the first part of our semester-long project towards creating an intelligent, multimodal assistant. In this part, we will be creating a command-line terminal where users can query in natural language. A command line interface is arguably the most poorly designed interface because the names of the commands are not intuitive and the number of parameters are somewhat arbitrary. As such, users must memorize the syntax of a command. It may work for expert users but not for regular computer users.

By completing this assignment, you will:

- Gain hands-on experience with how to use local, large, language model to create LLM-powered app
- How to create an agentic interaction with tool usage and MCP protocol
- Learn how to wrap command-line operations as callable tools in a server.
- Use of streaming model outputs and maintaining history to support deictic conversation.
- Explore how to handle tool results in a conversational setting.

Project Description

You will implement an MCP-based system consisting of:

1. **MCP Server (Terminal Tool)**
   - Build a server that exposes terminal-related operations as MCP tools:
     - `initiate_terminal`: starts a persistent shell session.
     - `run_command`: executes the given bash command within the terminal and returns its output.
     - `terminate_terminal`: safely closes the shell session.
   - Ensure the server maintains state (e.g., directory changes) across commands.
   - Handle process I/O properly using Python's subprocess.
   - **Example:**
     - Client: `cd ~/Desktop`, Tool: (changes directory inside the terminal session)
     - Client: `ls`, Tool: returns list of files in ~/Desktop
2. **MCP Client (Conversational Agent)**
   - Build a client that connects to the MCP server.

- Use Ollama as the underlying LLM (gpt-oss:20b or llama3.2:3b)
- Maintain a loop for the conversation where the user enters either natural language or direct commands.
- The model should:
  - Translate natural requests into commands (e.g., “What’s in this folder?” → `ls -la`).
  - Verify direct commands before sending to the tool.
  - Summarize terminal outputs into human-readable explanations.
- **Example:**
  - User: What files are in this directory?
  - Model: decides which tool to use and the command → `call_tool(tool_name="run_command", argument="ls -la")`
  - Tool: returns terminal output
  - Model: explains → "There are 3 files: assignment.pdf, notes.txt, and data.csv."

Here is how the system should work:

Starter Code

You will be provided with starter scripts to help you. The two scripts below are included only as demonstrations of how an MCP server and client are built.

- **weather_server.py (https://psu.instructure.com/courses/2416221/files/182366650?wrap=1)**  **(https://psu.instructure.com/courses/2416221/files/182366650/download?download_frd=1)** : This code implements an MCP server that provides tools to fetch and format weather alerts and forecasts from the U.S. National Weather Service (NWS) API.
- **weather_client.py (https://psu.instructure.com/courses/2416221/files/182366639?wrap=1)**  **(https://psu.instructure.com/courses/2416221/files/182366639/download?download_frd=1)** : This code implements an MCP client that connects to a weather tool server, uses an Ollama model to interpret

user queries, invokes the weather tools when needed, and returns natural-language responses about forecasts and alerts in an interactive chat loop.

Note: Their use case (weather information) is different from ours. These demos originate from the **MCP website's documentation (https://modelcontextprotocol.io/docs/develop/build-server)** . They have been slightly modified to work with the Ollama model, whereas the original documentation used Claude.

You are also provided with the server script for our use case, which implements MCP tools for terminal use. The client script will be written by you as part of the assignment.

- **terminal_server.py (https://psu.instructure.com/courses/2416221/files/182366653?wrap=1) (https://psu.instructure.com/courses/2416221/files/182366653/download?download_frd=1)** : This code implements an MCP server that provides tools to start, run commands in, and terminate a persistent terminal session.

References/Documentation

- Whiteboard (system workflow): **https://zoom.us/wb/doc/lJdR2538TNyf56f60rZkPw/p/239991448010752 (https://zoom.us/wb/doc/lJdR2538TNyf56f60rZkPw/p/239991448010752)**
- MCP documentation for building the server: **https://modelcontextprotocol.io/docs/develop/build-server (https://modelcontextprotocol.io/docs/develop/build-server)**
- MCP documentation for building the client: **https://modelcontextprotocol.io/docs/develop/build-client (https://modelcontextprotocol.io/docs/develop/build-client)**
- Ollama documentation for using the tool: **https://ollama.com/blog/tool-support (https://ollama.com/blog/tool-support)**
- Ollama documentation for streaming the model response: **https://ollama.com/blog/streaming-tool (https://ollama.com/blog/streaming-tool)**

Submission

A zip file containing:

- Source code
- A demo video/screencast of you using the system

Good luck, and enjoy building your intelligent conversational agent!