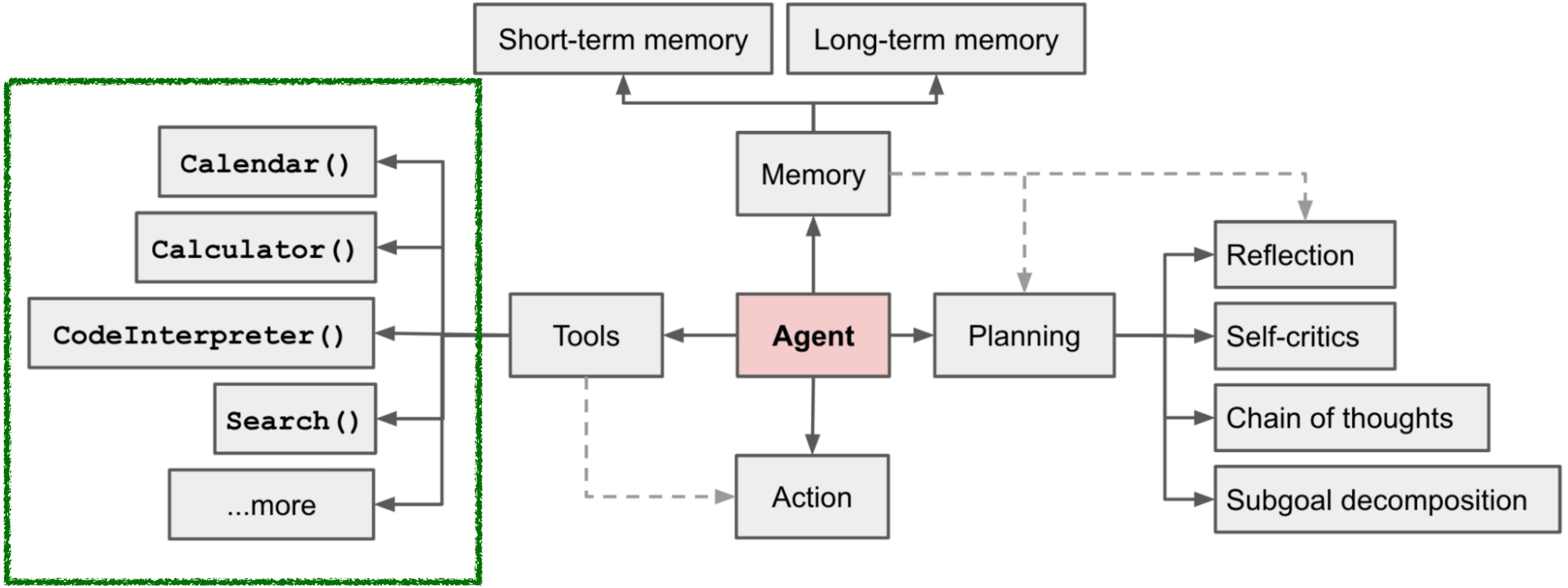
CS454

MCP Assignment Tutorial

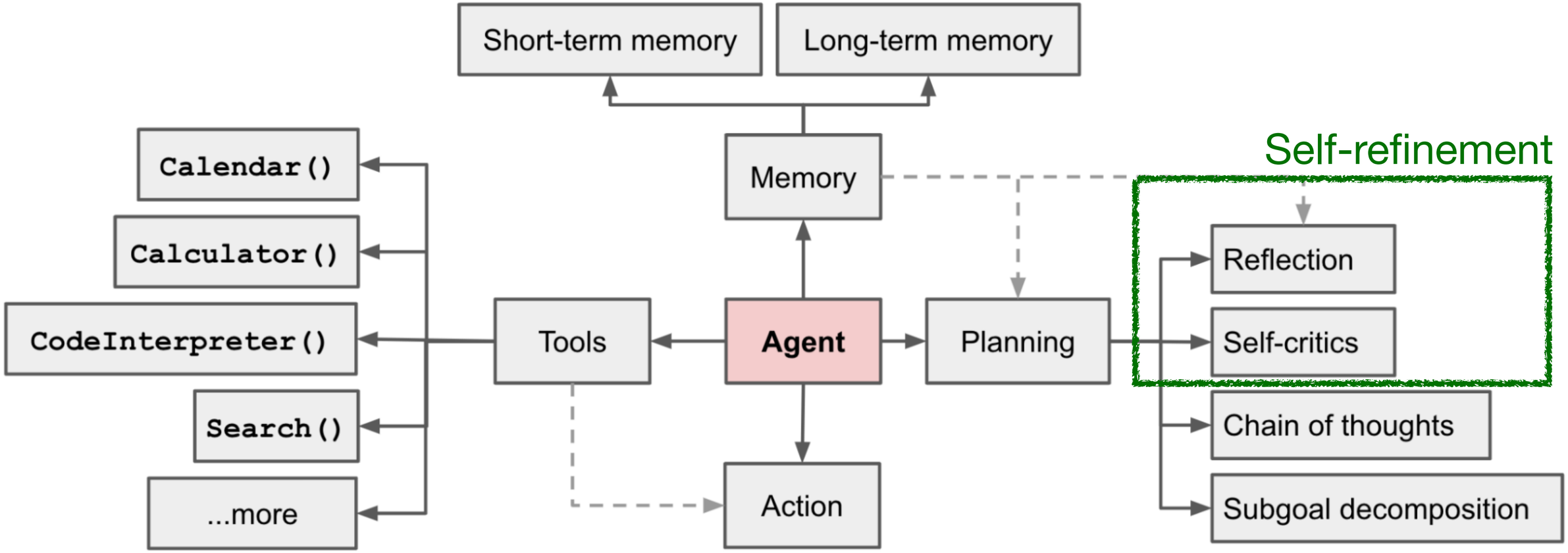
2025. 12. 04 THU

Building LLM Agents

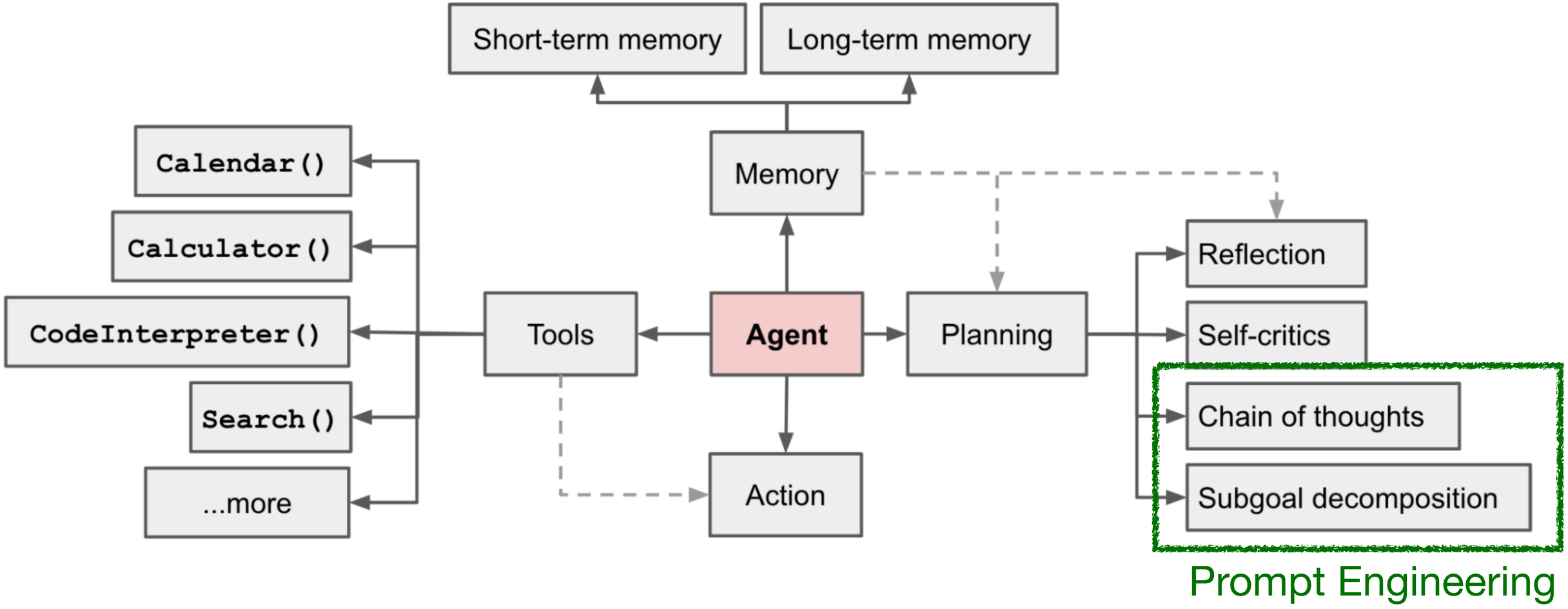


Tool Calls

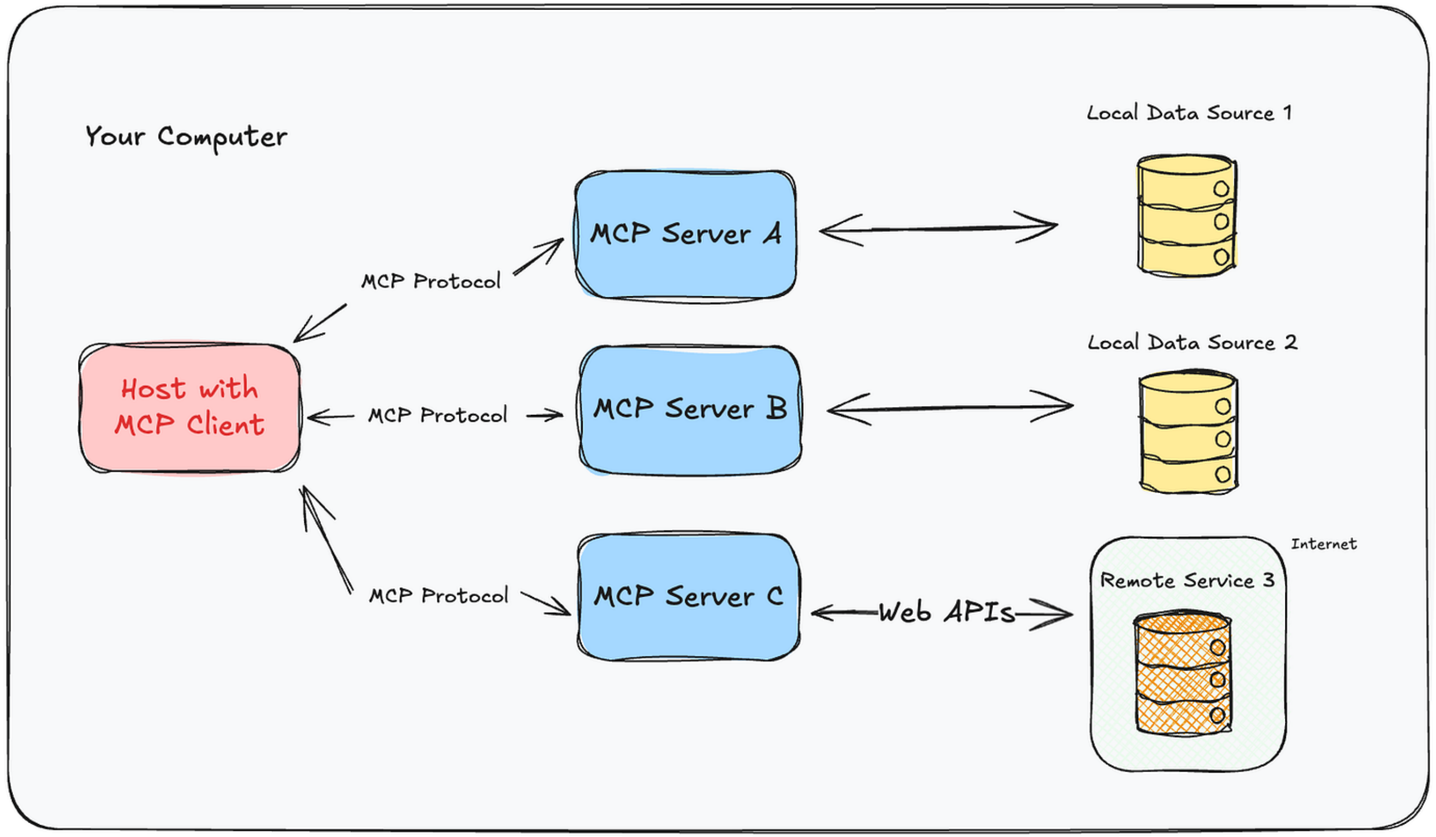
Building LLM Agents

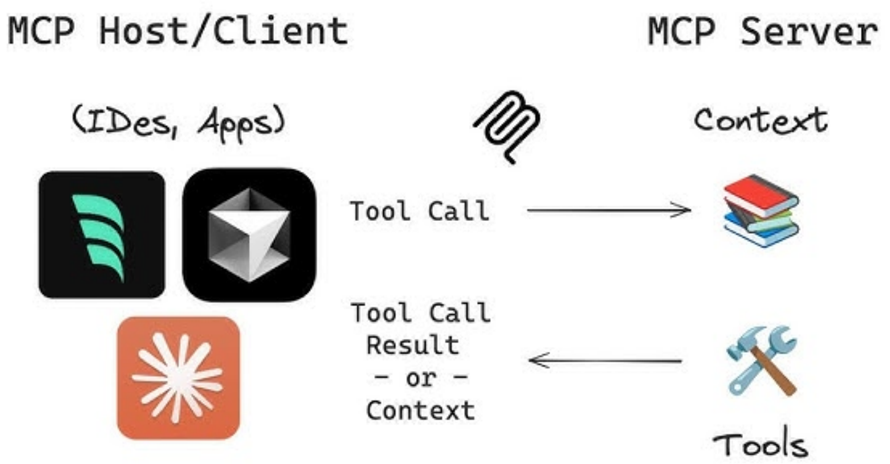


Building LLM Agents



MCP (Model Context Protocol)





Tutorial: Use Playwright MCP Server

- Environment Setup (Requires Node.js version >18):
  - https://nodejs.org/en/download
- If you have Claude Desktop / Copilot Chat / Cursor installed:

ex) claude_desktop_config.json:

```
"playwright": {
      "command": "npx",
      "args": [
        "@playwright/mcp@latest"
      ]
    }
```

Environmental Setup (Python mcp lib)

```
> python -m venv .venv

> source .venv/bin/activate (Windows: .venv\Scripts\activate)

> pip install mcp openai
```

Connect to Playwright MCP

```
async def main(server_script_path: str):
    client = MCPClient()
    try:
        # await client.connect_to_server("python", [server_script_path])
        goal: run bash command (npx -y @playwright/mcp@latest —output-dir ./)
        await client.chat_loop()
    finally:
        await client.cleanup()
```

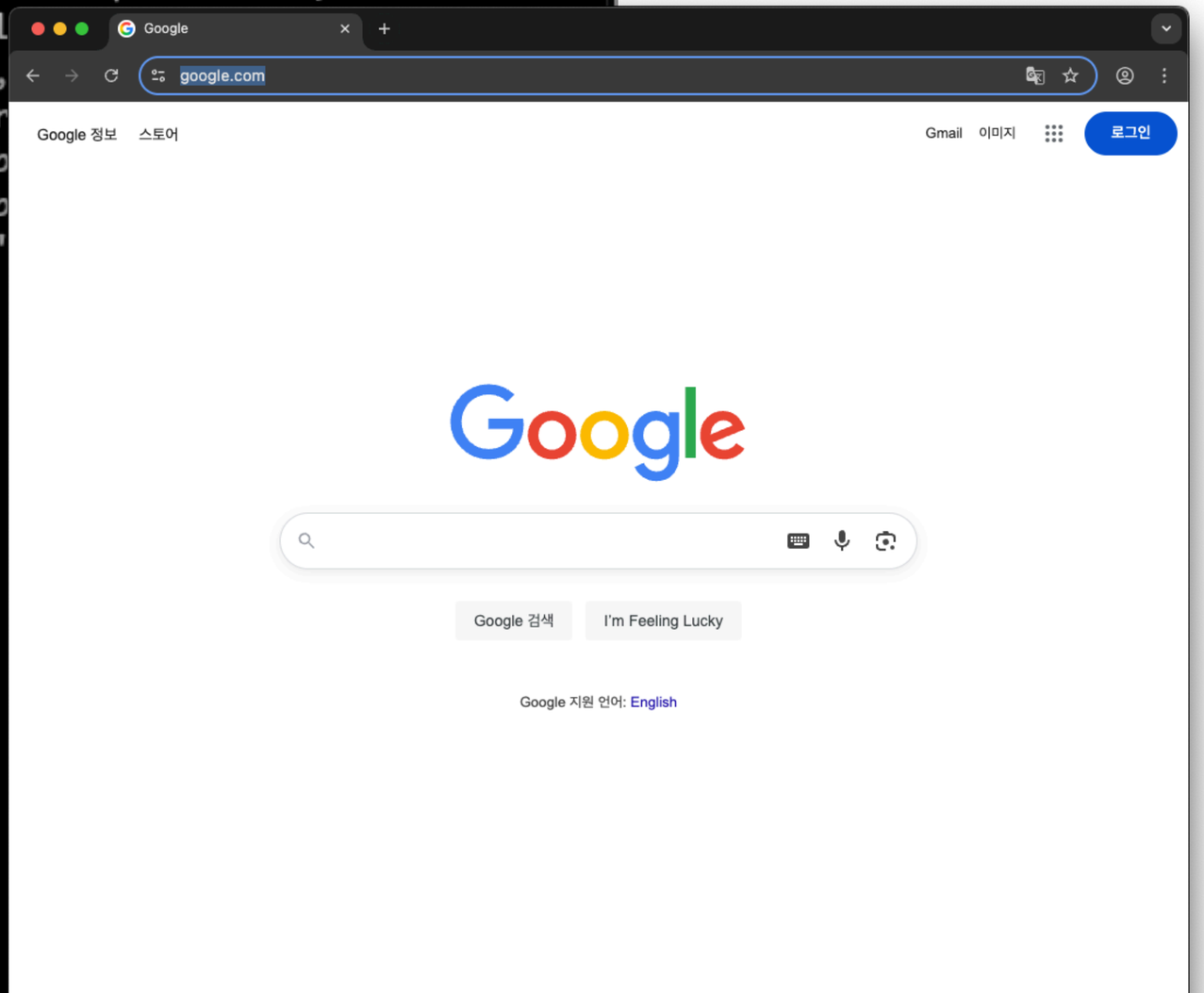
Connect to Playwright MCP

```
async def main(server_script_path: str):
    client = MCPClient()
    try:
        # await client.connect_to_server(“python”, [server_script_path])
        await client.connect_to_server(
            command="npx",
            args=["-y", "@playwright/mcp@latest", "--output-dir", "./"]
        )
        await client.chat_loop()
    finally:
        await client.cleanup()
```

Requesting web testing to the agent

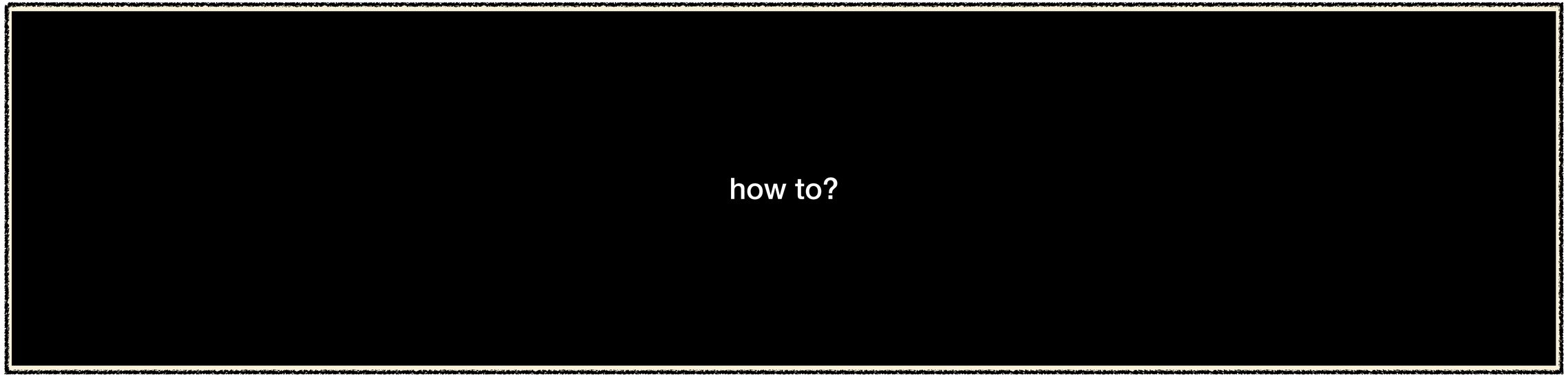
```
Connected to server (npx -y @playwright/mcp@latest --output-dir ./) with tools:
['browser_close', 'browser_resize', 'browser_console_messages', 'browser_handle_
dialog', 'browser_evaluate', 'browser_file_upload', 'browser_install', 'browser_
press_key', 'browser_type', 'browser_navigate', 'browser_navigate_back', 'browse
r_navigate_forward', 'browser_network_requests', 'browser_take_screenshot', 'bro
wser_snapshot', 'browser_click', 'browser_drag', 'browser_hover', 'browser_selec
t_option', 'browser_tab_list', 'browser_tab_new', 'browser_tab_select', 'browser
_tab_close', 'browser_wait_for']
Welcome to the MCP Client! Type 'exit' to quit.

You: google.com이 잘 동작하는지 테스트하고 싶어

Agent:

* `url`:
```
https://www.google.com
```
* `browser_navigate`:
```
Ran Playwright code
```js
await page.goto('https://www.google.com');
```
```
```

Requesting web testing to the agent

```
Connected to server (npx -y @playwright/mcp@latest --output-dir ./) with tools:
['browser_close', 'browser_resize', 'browser_consol
dialog', 'browser_evaluate', 'browser_file_upload',
press_key', 'browser_type', 'browser_navigate', 'br
r_navigate_forward', 'browser_network_requests', 'b
wser_snapshot', 'browser_click', 'browser_drag', 'b
t_option', 'browser_tab_list', 'browser_tab_new', '
_tab_close', 'browser_wait_for']
Welcome to the MCP Client! Type 'exit' to quit.

You: google.com이 잘 동작하는지 테스트하고 싶어

Agent:

* `url`:
```
https://www.google.com
```
* `browser_navigate`:
```
Ran Playwright code
```js
await page.goto('https://www.google.com');
```
```


```

Postprocessing Tool Return Values

- amazon.com → Request testing “Search” functionality
  - gpt-4o-mini max context length (~128000 tokens)
  - **Quick Fix:** truncate text (+ better w/ *tiktoken*, summarization)



Chat-based “Assistant” to “Agent”

```
async def testing_loop(self, target_websites):
        print(f"Testing Agent for the websites: {target_websites}")
        self.messages = [{
            "role": "system",
            "content": "You are a testing agent that can interact with
web pages and perform tasks."
        }]

        await self.process_tool_call({
            "type": "function",
            "function": {
                "name": "browser_navigate",
                "arguments": json.dumps({
                    "url": target_websites
                })
            },
            "id": "initial_navigation"
        })
```

Custom Prompting

Tune the default task instruction

```
while True:
    user_input = input("\nProvide a task instruction for testing (or 'exit' to quit): ")
    if user_input.lower() == 'exit':
        break

    while True:
        self.messages.append(process_task_instruction(user_input, target_websites))

        try:
            self.messages = await self.process_messages_streaming(self.messages)
```

Perform the following task (in https://sciencedirect.com)

Find how many articles about "renewable energy" were published by Harvard from the last 5 years, filter by open access

After completing it, please provide a brief testing report.

Tutorial 2 (Assignment 5): Build PyTest MCP

- *PlayWright MCP "server"* is developed by Microsoft, to be compatible with any clients that implement the model context protocol (e.g., VSCode Copilot Chat, Claude Desktop, Cursor)

- Assignment 4 is basically about building a custom **MCP server,** and tuning the client-side logic.

Tool: list_files()

tools-task-1.py

```
from mcp.server.fastmcp import FastMCP

mcp = FastMCP(“pytest-automator")

target_directory = os.path.join(os.path.dirname(__file__), “target_programs”)


@mcp.tool()
def list_files() -> str:
    """
    List all Python files in the target directory to generate tests for.
    """
    # TODO
```

`@mcp.tool()`: Add a tool to MCP client

Tool: list_files()

tools-task-1.py

```
@mcp.tool()
def list_files() -> str:
    """
    List all Python files in the target directory to generate tests for.
    """
    file_list = []
    for file_path in glob.glob(os.path.join(TARGET_DIR, "*/*.py")):
        file_list.append(file_path.removeprefix(TARGET_DIR + os.sep))
    try:
        files = "\n".join(file_list)
        return f"All Python files in the target directory:\n{files}"
    except Exception as e:
        return f"Error listing files: {str(e)}"
```

`@mcp.tool()`: Add a tool to MCP client

MCP Client

- Basically, Copilot, Claude require chat-based user input
- What if I want to create a user interface on my own?
  - ex) When I type ‘TEST_GEN’, I want the agent to begin on the test generation loop
  - Create a custom client!

MCP Client

```
class MCPClient:
    def __init__(self):
        # Initialize session and client objects
        self.session: Optional[ClientSession] = None
        self.exit_stack = AsyncExitStack()
        self.llm = OpenAI()


    async def connect_to_server(self, server_script_path: str):
        server_params = StdioServerParameters(command="python", args=[server_script_path], env=None)

        stdio_transport = await self.exit_stack.enter_async_context(stdio_client(server_params))
        self.stdio, self.write = stdio_transport
        self.session = await self.exit_stack.enter_async_context(ClientSession(self.stdio,
self.write))

        await self.session.initialize()

        # List available tools
        response = await self.session.list_tools()
        tools = response.tools
```

“Guardrails” for Agent Execution

- LLM can make *mistakes*:
  - Wrong file name when reading / writing
  - Forgetting to add a “.py”
  - Writing ‘nothing’ to a file…
- Implement guardrails!

```
@mcp.tool()
def read_file(filename) -> str:
    if ‘.py’ not in filename:
```

Task 1: Implement Basic MCP Server

- Just a running MCP server
- Tools you need to implement:
  - `list_files()`
  - `read_files()`
  - `write_file()`
- Implement MCP client in mcp-client-task-1.py
- 30% of the test score

Task 2: Automated Test Generation

- Use LLM to generate tests automatically!
- Tools you need to implement:
  - `run_pytest()`: Run pytest on the given Python test file
  - `measure_coverage()`: Measure code coverage
- 50% of the test score

Task 3: Call SBST

- Make LLM agent to run your SBST implementation
- Tools you need to implement:
  - `run_sbst()`: Run your implementation in sbst.py
- 20% of the test score
- What if I did not do assignment 1?
  - Implement a ‘sbst.py’ that generates random input
  - However, maximum 10% contribution (-10% deduction from 20% above)

Important: $ 3.00 distributed for each of you!

- Even if you run out, we designate you more budget
- However, our implementation costed us **$0.003 ~ $0.04**: 100~1000 calls
- You can estimate your token usage by referring to OpenAI API docs
- You can try using Qwen3 (or other OLMs) through Ollama

- **Make sure to follow all input-output formats described in README.md**