Chapter 6 - Future outlook & Clsoing

Intro

Hello everyone, welcome back!

In the previous video, we learned how to integrate a Large Language Model (LLM) with the MCP (Model Context Protocol) tool, along with some associated security considerations.

Today, we're wrapping up our MCP basics series. In this final video, we'll cover:

1. What we didn't cover in the previous videos and why.
2. The future outlook of MCP and related approaches.

Let's dive in!

What We Didn't Cover and Why

So far, in this series, we've explored many aspects of MCP:

- Why MCP was developed, by understanding the concept of tools and how AI models can call them.
- Basic MCP concepts such as clients and servers.
- How to create an MCP tool, call it, and connect it to an existing MCP server.
- How to integrate MCP tools with Large Language Models.

However, even in this final video, there are still some topics we didn't touch. Before closing the courses, I'd like to let you know one important concept, **resources**.

A **resource** is similar to a tool, but there's a key difference: it doesn't perform significant computations. You can think of it more like a structured piece of data.

Example:

```
@mcp.resource("file://{path}")
def read_file(path: str) → str:
```

```
    """
    Return the contents of the file at the given path (read-only).
    """
    with open(path, 'r', encoding='utf-8') as f:
        return f.read()
```

According to the official documentation, resources are how you expose data to LLMs—they provide information without performing major computations. Conceptually, this makes them quite different from tools.

So why didn't we cover everything?

1. **Focus on the essentials:** For beginners, it's easier to start with the most important concepts. Covering too much at once can be overwhelming—like getting lost in a forest.
2. **You're ready to explore further:** By now, you understand most of the core MCP concepts. This foundation makes it easier to read the official documentation or explore advanced topics on your own.

Future Outlook

Let's recap how traditional MCP tool calling works:

1. The LLM generates a JSON-formatted output containing the tool name and arguments.
2. The MCP client sends this JSON to the MCP server using the JSON-RPC protocol.
3. The server executes the tool and sends the results back to the client.
4. The MCP client inserts these results into the LLM's context.
5. The LLM continues generating output based on this updated context.

This method works well, but it's not the only way to call a tool.

A newer approach is **CodeAct**, which leverages the LLM's ability to generate executable code directly.

Here's the idea: instead of relying on human-defined tools, the LLM generates Python code to solve a task.

For example:

User:

Please analyse the CSV file located at ./data.csv

CodeAct Agent:

To accomplish this task, I will use the pandas library....

```
import pandas as pd
import numpy as np

# Load dataset
df = pd.read_csv(...)

# Check for missing values
print(df.isnull().sum())

# Further processing...
```

The above generated code is executed by a python interpreter, and the results are fed back into the LLM's context. The LLM then generates the next steps to complete the task.

Interestingly, the authors of CodeAct suggest that this approach can outperform traditional JSON-based tool calling. Frameworks like **LangGraph** and Hugging Face's **agent library** are starting to support CodeAct-style agents.

One important note: MCP is built around JSON-RPC to standardise tool calls, whereas CodeAct may require a different protocol since it doesn't require JSON formatting for tool calls. Instaed, it requires python interpreter.

Actually, tool calling is still an active area of research. We're only just beginning to explore how to use tools with LLMs effectively, and in the future, the methods and standards could look quite different from what we use today.

For more information:

https://arxiv.org/abs/2402.01030

https://huggingface.co/learn/agents-course/unit2/smolagents/code_agents

---

Recap

To summarise:

- We've covered all the core MCP concepts you need to get started.

- Looking forward, LLMs are evolving to handle code execution tasks directly, as seen in CodeAct, which could influence future MCP-like frameworks.

That’s it for the MCP basics series! Thank you for following along, and I hope you’re ready to explore advanced MCP topics and other next-generation LLM tools.