LangChain MCP

(Model Context Protocol)

Simple Guide for Students

What is MCP?

MCP (Model Context Protocol) allows AI models to interact with external tools like file systems, databases, APIs, etc. Think of it as giving AI 'hands' to do things in the real world.

How It Works - Simple Explanation

1. Setup MCP Client

```
client = MultiServerMCPClient({
    "filesystem": {
        "transport": "stdio",
        "command": npx_command(),
        "args": ["-y", "@modelcontextprotocol/server-filesystem",
                 "C:\\Users\\shram\\mcp"]
    }
})
```

What it does: Creates a connection to the MCP filesystem server that can read/write files in C:\Users\shram\mcp

2. Get Available Tools

```
tools = await client.get_tools()
```

What it does: Asks the MCP server 'what can you do?' and gets back 14 tools:
• read_file - Read file contents
• write_file - Create/write files
• list_directory - List files in folder
• create_directory - Make new folders
• and 10 more tools...

3. Connect LLM with Tools

```
llm = ChatOpenAI(model="gpt-4o", api_key=os.getenv("OPENAI_API_KEY"))
llm_with_tools = llm.bind_tools(tools)
```

What it does: Gives GPT-4o access to all 14 filesystem tools

4. The Chat Loop - How AI Uses Tools

```
messages = []
messages.append(SystemMessage(content=f"All file operations are in: {MCP_FOLDER}"))
messages.append(HumanMessage(content=query))

while True:
    ai_msg = llm_with_tools.invoke(messages)
    messages.append(ai_msg)

    if not ai_msg.tool_calls:
        return ai_msg.content

    for tc in ai_msg.tool_calls:
        tool_name = tc["name"]
        tool_args = tc["args"]
        tool_fn = next(t for t in tools if t.name == tool_name)
        tool_result = await tool_fn.ainvoke(tool_args)
        messages.append(ToolMessage(content=str(tool_result),
                                    tool_call_id=tc["id"]))
```

Why Async is Needed?

Understanding Async/Await

MCP uses **async/await** because it involves I/O operations (Input/Output) like:
• Reading/writing files
• Network communication with MCP servers
• API calls to OpenAI
• Process communication (stdio)

The Problem Without Async

Imagine this scenario:

```
# Synchronous (blocking) code
file1 = read_file("file1.txt")     # Wait 2 seconds
file2 = read_file("file2.txt")     # Wait 2 seconds
file3 = read_file("file3.txt")     # Wait 2 seconds
# Total time: 6 seconds (each waits for previous)
```

With synchronous code, your program **blocks** (freezes) while waiting for each operation. Nothing else can happen during this time.

The Solution With Async

```
# Asynchronous (non-blocking) code
file1 = await read_file("file1.txt")   # Start, then can do other work
file2 = await read_file("file2.txt")   # Start, then can do other work
file3 = await read_file("file3.txt")   # Start, then can do other work
# Total time: ~2 seconds (can run in parallel)
```

With async code, while waiting for one operation, Python can work on other tasks. The program doesn't freeze.

Real-World Analogy

Synchronous (Blocking):
You're at a restaurant. You order food, then stand at the counter waiting until it's ready. You can't do anything else while waiting. If 3 people order, they all wait in line - one after another.

Asynchronous (Non-Blocking):
You're at a restaurant. You order food, get a number, and sit down. While the kitchen prepares your food, you can read, work, or talk. When ready, they call your number. Multiple orders can be prepared at the same time.

In MCP Code

```
# Without async - This would BLOCK
def chat(query):
    client = MultiServerMCPClient(...)
    tools = client.get_tools()  # Program freezes here
    # Can't do anything until tools are loaded
```

```
# With async - Program stays responsive
async def chat(query):
    client = MultiServerMCPClient(...)
    tools = await client.get_tools()  # Can do other work while waiting
    # Program continues, can handle other requests
```

Key Async/Await Rules

Keyword	Purpose	Example
async def	Define async function	async def chat(): ...
await	Wait for async operation	result = await tool.ainvoke()
asyncio.run()	Run async function	asyncio.run(chat("hello"))

Important Notes

• You **cannot** use 'await' outside an 'async' function
• To call async function from normal code, use: **asyncio.run()**
• MCP operations use 'await' because they involve I/O (waiting for external processes)
• 'ainvoke' means 'async invoke' - it's the async version of 'invoke'

Why MCP Needs Async

MCP communicates with external servers via **stdio** (standard input/output):

```
User Query
    ↓
Your Python Code (async)
    ↓
MCP Client (async communication)
    ↓
NPX Process (filesystem server)
    ↓
File System Operations
    ↓
Results back to Python
```

Each arrow (↓) involves waiting for I/O. Using async means your program can handle multiple requests while waiting for these operations.

Step-by-Step Example

User asks: 'create a file named python.txt with content Hello, World!'

Step 1: AI Analyzes Request

• AI thinks: 'I need to create a file'
• AI decides: 'I'll use the write_file tool'

Step 2: AI Calls Tool

```
{
  "name": "write_file",
  "args": {
    "path": "C:\\Users\\shram\\mcp\\python.txt",
    "content": "Hello, World!"
  }
}
```

Step 3: Tool Executes

• MCP server creates the file
• Returns: 'File created successfully'

Step 4: AI Responds to User

'I've created the file python.txt with the content Hello, World!'

Message Flow Diagram

User:	create a file named python.txt
↓	
SystemMessage:	All file operations are in: C:\Users\shram\mcp
HumanMessage:	create a file named python.txt
↓	
AI (GPT-4o):	I'll use write_file tool
↓	
ToolCall:	write_file(path="...", content="Hello, World!")
↓	
MCP Server:	Executes → Creates file
↓	
ToolMessage:	File created successfully
↓	
AI (GPT-4o):	I've created the file for you!
↓	
User:	Gets response

Key Concepts

1. SystemMessage

```
SystemMessage(content=f"All file operations are in: {MCP_FOLDER}")
```

- Sets rules/context for AI
- Tells AI where files should be created
- AI follows this instruction for all operations

2. HumanMessage

```
HumanMessage(content=query)
```

- Your question/request
- Example: 'list files', 'create file', 'read file'

3. ToolMessage

```
ToolMessage(content=str(tool_result), tool_call_id=tc["id"])
```

- Result from tool execution
- AI sees this and can respond or call more tools

4. The Loop

- AI can call multiple tools in one request
- Loop continues until AI has final answer
- Example: 'copy file A to B' might need: read_file → write_file

Simple Test Examples

List files:

```
response = asyncio.run(chat("List all files"))
print(response)
```

Create file:

```
response = asyncio.run(chat("create a file named test.txt with content 'Hello'"))
print(response)
```

Read file:

```
response = asyncio.run(chat("read the content of test.txt"))
print(response)
```

Create directory:

```
response = asyncio.run(chat("create a new directory named MyFolder"))
print(response)
```

Common Errors & Solutions

Error	Solution
Access denied - path outside allowed directories	Add SystemMessage with correct folder path
Tool not found	Check tool_name matches available tools
Hanging/No response	Add max iterations limit or better prompts

Complete Working Code

```
import asyncio
import os
import platform
from dotenv import load_dotenv
from langchain_openai import ChatOpenAI
from langchain_mcp_adapters.client import MultiServerMCPClient
from langchain_core.messages import HumanMessage, ToolMessage, SystemMessage

load_dotenv()

MCP_FOLDER = r"C:\Users\shram\mcp"

def npx_command():
    return r"C:\Program Files\nodejs\npx.cmd" if platform.system().lower().startswith("win") else "npx"

async def chat(query: str) -> str:
    # 1. Create MCP client
    client = MultiServerMCPClient({
        "filesystem": {
            "transport": "stdio",
            "command": npx_command(),
            "args": ["-y", "@modelcontextprotocol/server-filesystem", MCP_FOLDER],
        }
    })

    # 2. Get available tools
    tools = await client.get_tools()

    # 3. Create LLM with tools
    llm = ChatOpenAI(model="gpt-4o", api_key=os.getenv("OPENAI_API_KEY"))
    llm_with_tools = llm.bind_tools(tools)

    # 4. Setup messages
    messages = []
    messages.append(SystemMessage(content=f"All file operations are in: {MCP_FOLDER}"))
    messages.append(HumanMessage(content=query))

    # 5. Chat loop - AI uses tools until done
    while True:
        ai_msg = llm_with_tools.invoke(messages)
        messages.append(ai_msg)

        if not ai_msg.tool_calls:
            return ai_msg.content

        for tc in ai_msg.tool_calls:
            tool_name = tc["name"]
            tool_args = tc["args"]
            tool_fn = next(t for t in tools if t.name == tool_name)
            tool_result = await tool_fn.ainvoke(tool_args)
```

```
            messages.append(ToolMessage(content=str(tool_result),
                                        tool_call_id=tc["id"]))

# Usage
response = asyncio.run(chat("List all files"))
print(response)
```