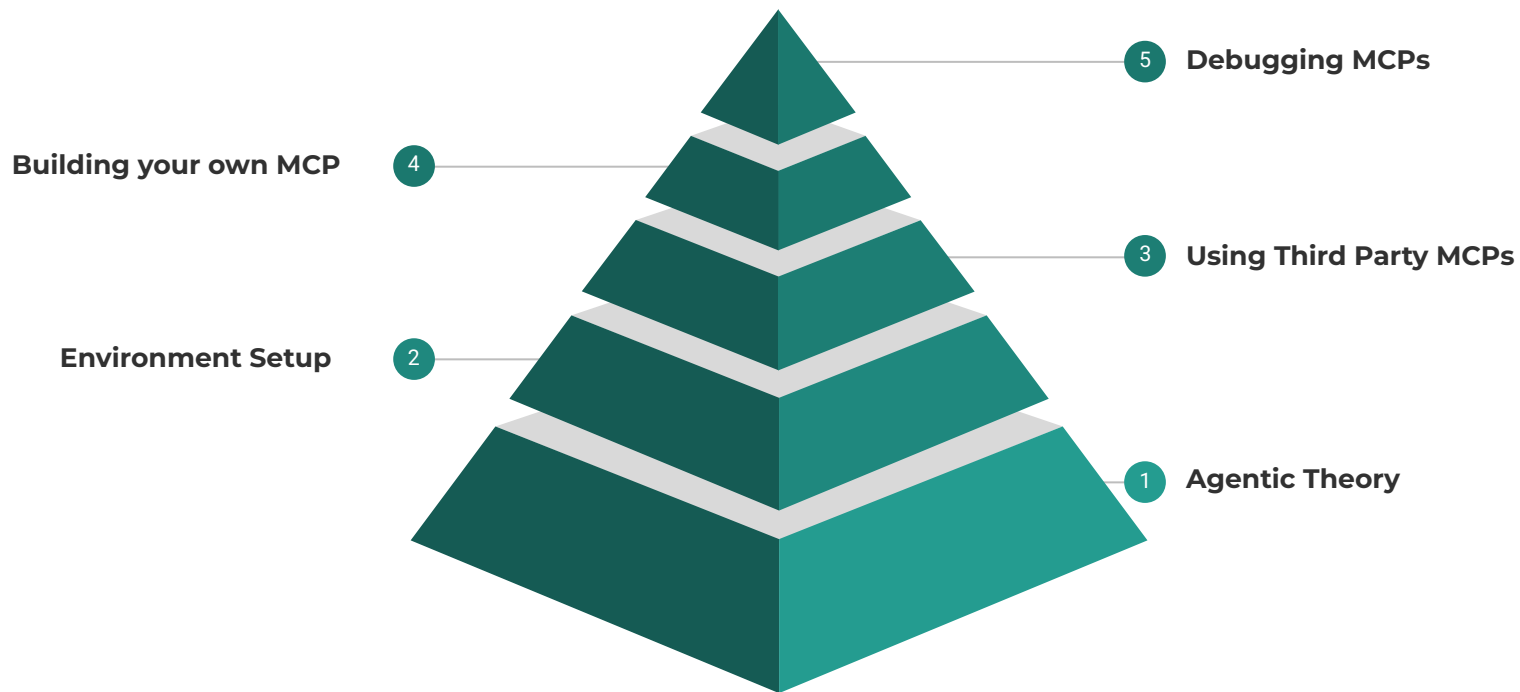


# The Complete MCP (Model Context Protocol) Bootcamp

<<Udemy link>>



# Course Blueprint



# How to make the most of this course

- Link to the **Github repository, Presentations and Discord** are linked the next Lecture
- The rest in **COURSE\_RESOURCES.md**

# Manage Speed and Section Completion

## COURSE BLUEPRINT

Implementing Chat History  
Awareness

Local LLM  
Hello World

5 Implementing a Chat UI

3 Adding Chat Capabilities to the LLM

1 Intro & Setting up our  
Environment

### Course content

#### Section 1: Introduction and Environment Setup

5 / 7 | 13min

- ☒ 1. Introduction
  - 📄 Resources
- ☒ 2. What will you learn?
  - 🕒 1min
- ☒ 3. Prerequisites and Requirements
  - 🕒 2min
- ☒ 4. Resources for Python, Github and LLM basics
  - 📄
- ☒ 5. Course Blueprint
  - 🕒 2min
- ☐ 6. How to take the most out of this course
  - 📄
- ☐ 7. Setting up the Coding Environment
  - 🕒 8min

#### Section 2: Getting a Local LLM to Work

0 / 5 | 17min

#### Section 3: Adding Chat Capabilities to the LLM

0 / 2 | 7min

#### Section 4: Implementing Chat History Awareness

0 / 1 | 6min

Playback rate

Nordquant

2:00 / 2:05

Overview Notes Announcements Reviews Learning tools

# Review

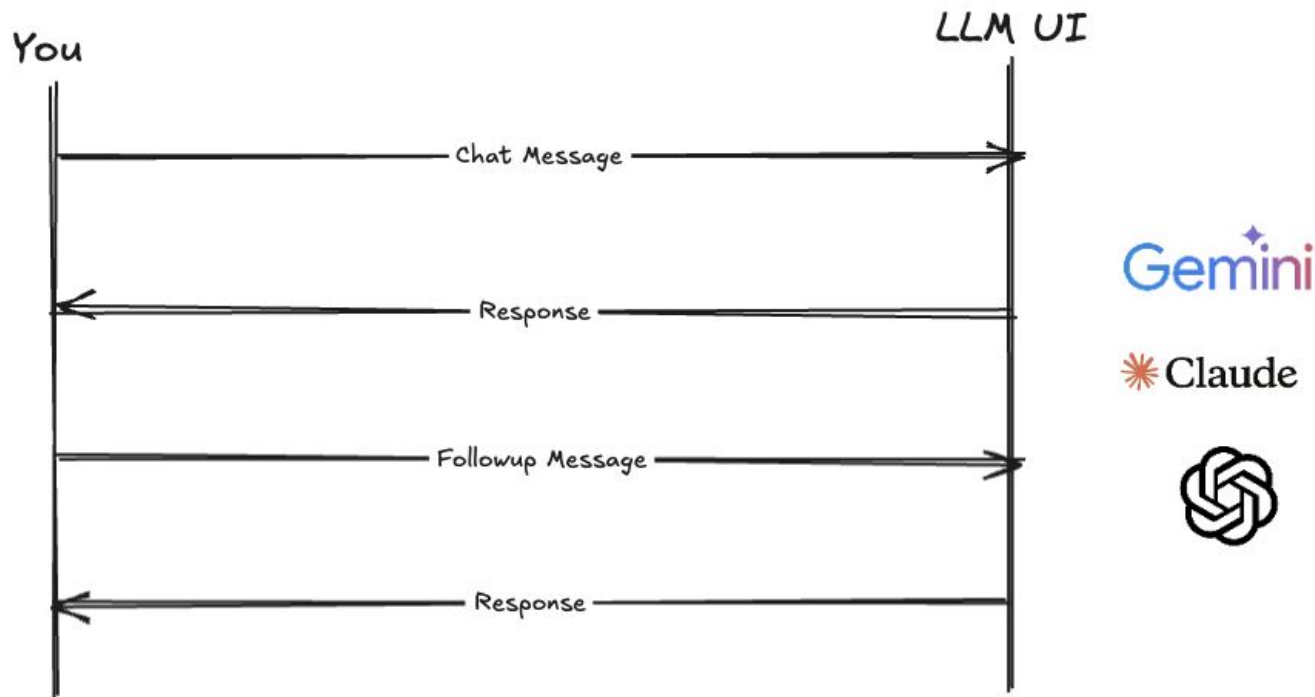


Ask Me Later

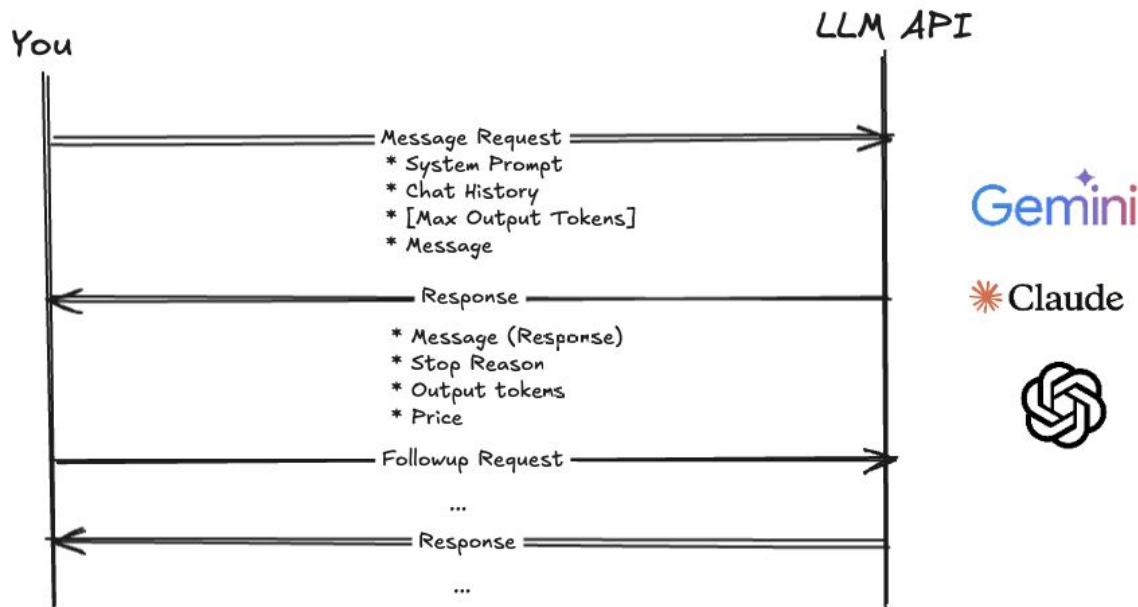
# Prerequisites

- Python
- Node JS
- Claude
- Cursor
- Visual Studio Code

# Interacting with an LLM through a UI



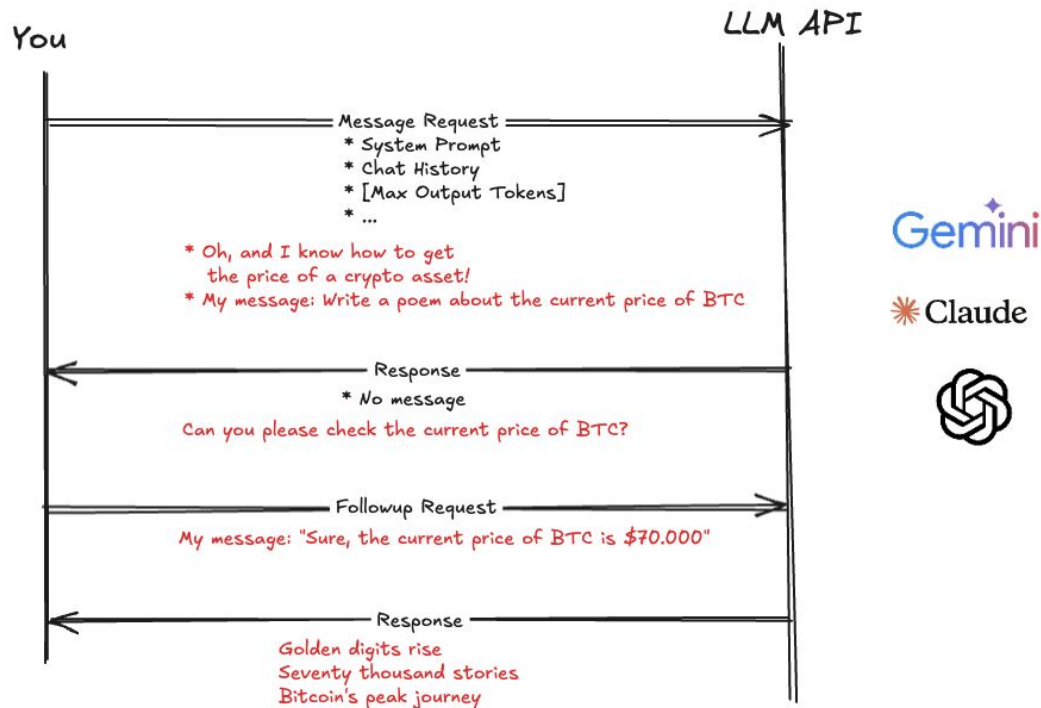
# LLM Interaction behind the scenes



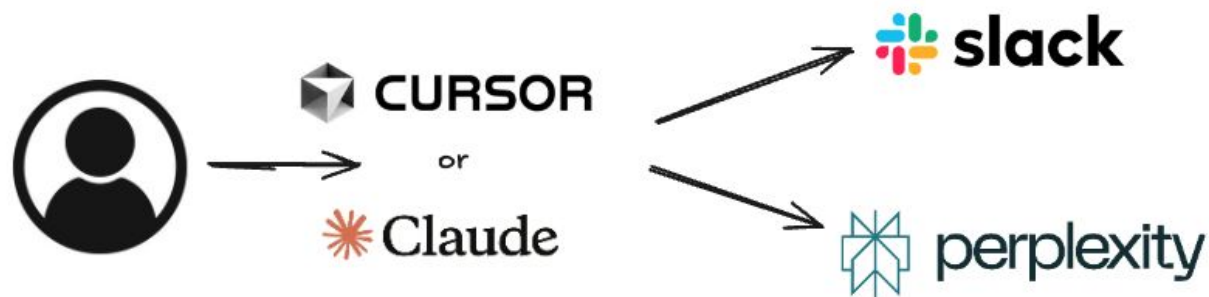


# LLM Tool Use

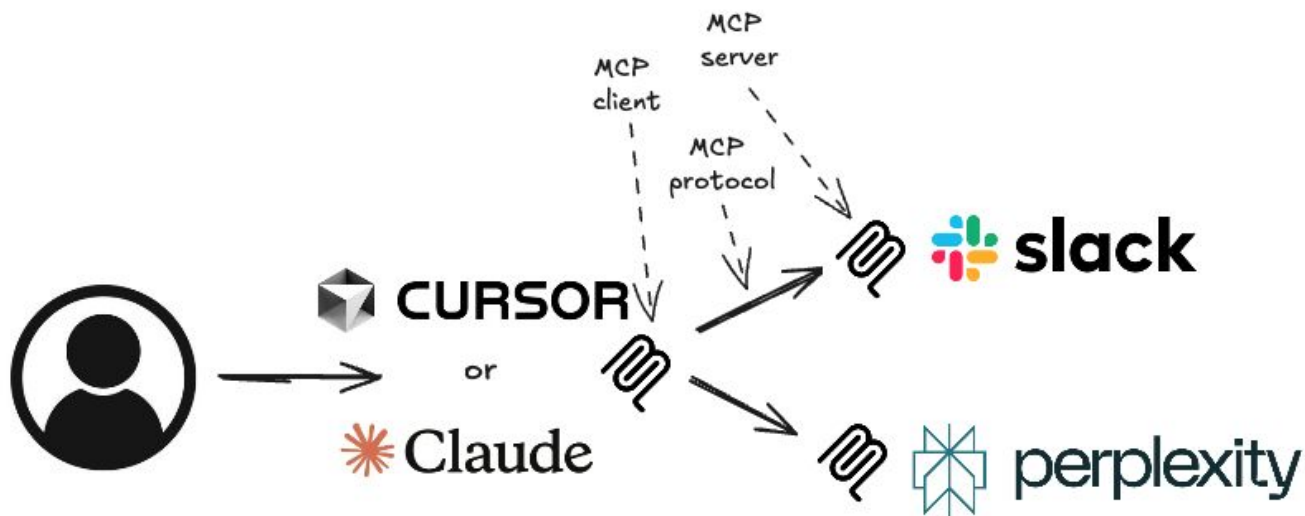
```
def get_crypto_price(symbol):  
    << get crypto price from a broker and return with the value>>
```



# The Problem MCP Solves



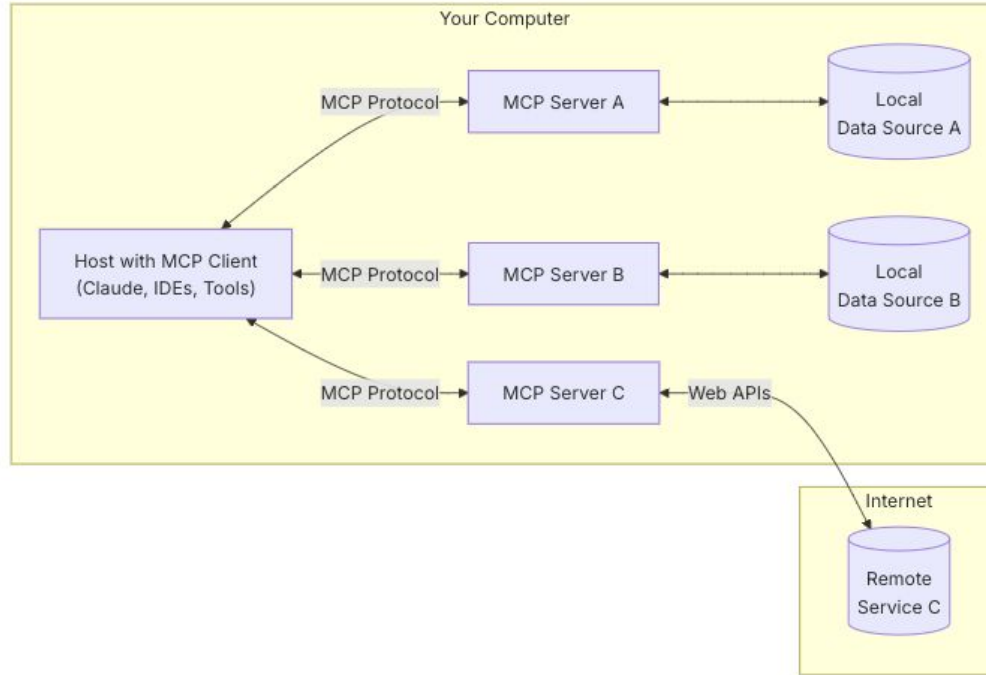
# MCP Architecture



# MCP Architecture Concepts

- **MCP Hosts:** Programs that want to access MCP Services, such as Claude or Cursor
- **MCP Protocol:** The language MCPs clients and servers use for communication and data passing
- **MCP Servers:** Server applications that expose functionalities for LLMs through the *MCP Protocol*
- **MCP Clients:** The client modules that maintain connections between an *MCP Host* and *MCP Server*

# MCP Architecture (continued)



# MCP functionalities

- **Tools:** “Tool Use”
- **Resources:** Providing the LLM with files and assets
- **Prompts:** Providing pre-created Prompts

Only partially supported features:

- **Roots:** Defining which resources to use with this MCP
- **Sampling:** Classic LLM next token prediction service

# Tools

```
mcp.tool()  
def run_command(command: str) -> str:  
    """  
    Execute a shell command and return its output  
  
    Args:  
        command (str): The shell command to execute  
  
    Returns:  
        str: The command's output  
    """  
    result = subprocess.run(command)  
    return result.stdout
```

# Resources

```
@app.list_resources()  
async def list_resources() -> list[types.Resource]:  
    return [  
        types.Resource(  
            uri="file:///opt/resources.md",  
            name="Course Resources",  
            mimeType="text/markdown"  
        )  
    ]  
  
@app.read_resource()  
async def read_resource(uri: AnyUrl) -> str:  
    if str(uri) == "file:///opt/resources.md":  
        with open("/opt/resources.md", "r") as file:  
            return file.read()  
  
    raise ValueError("Resource not found")
```



# Prompts

```
PROMPTS = {  
  "code-review": types.Prompt(  
    name="code-review",  
    description="""Review code for best practices, potential issues, and improvements.  
                    Focus on the core logic. Keep your answer simple. Use bulletpoints.  
                    Keep in mind that I am an experienced developer""",  
    arguments=[  
      types.PromptArgument(  
        name="code",  
        description="Code to review",  
        required=True  
      )  
    ]  
  )  
}
```





