Connecting C++ Tools to AI Agents Using the Model Context Protocol (MCP)

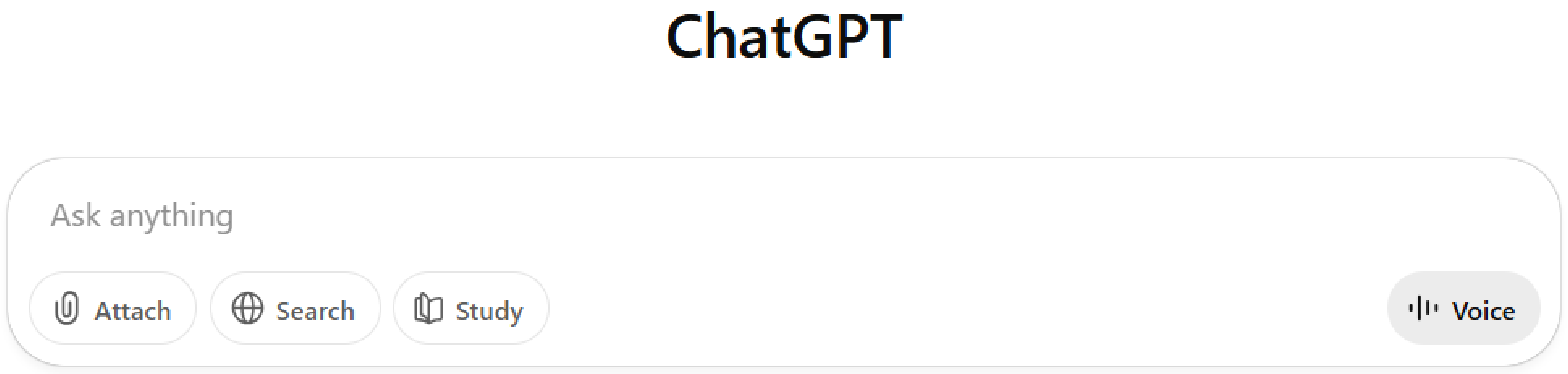
BEN MCMORRAN

Cppcon
The C++ Conference

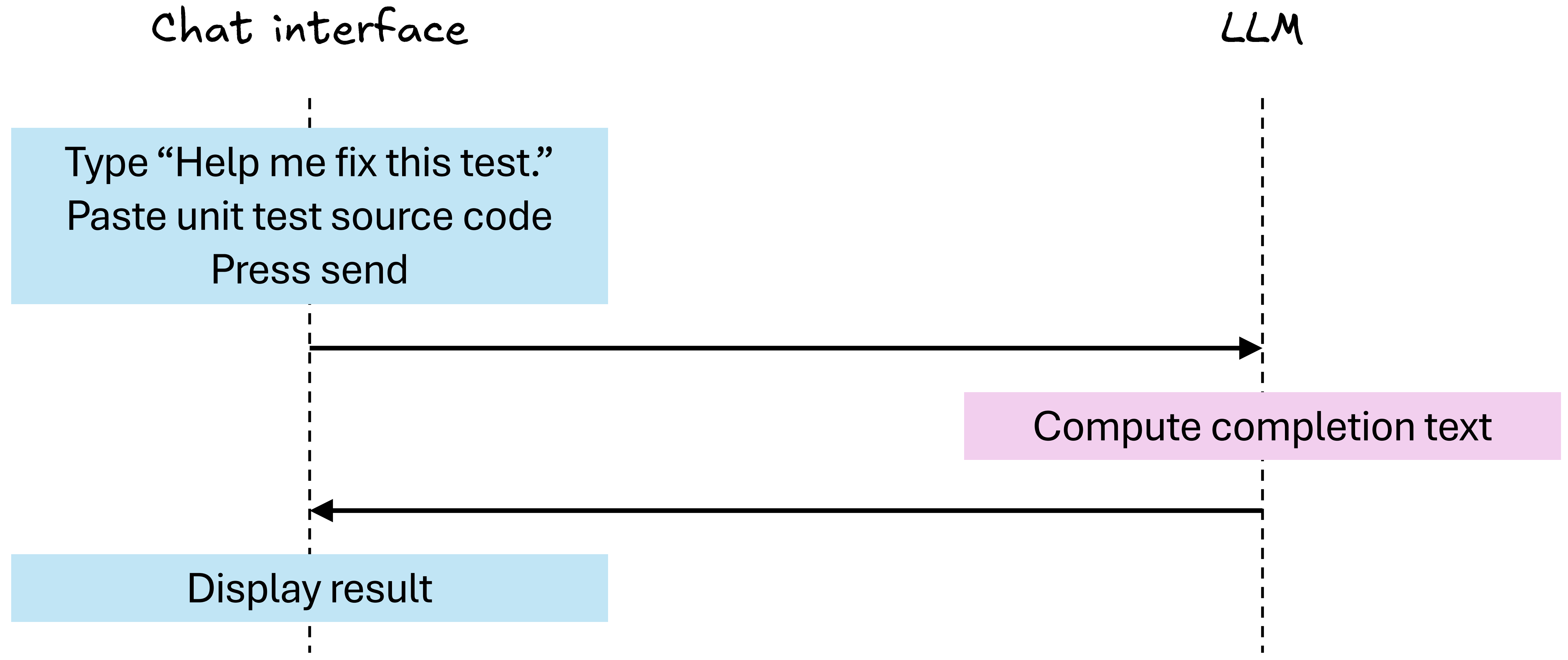
2025
September 13 - 19

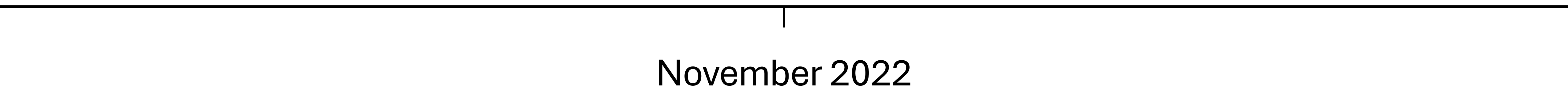
A brief, incomplete history of LLMs in developer tools

Chat is born

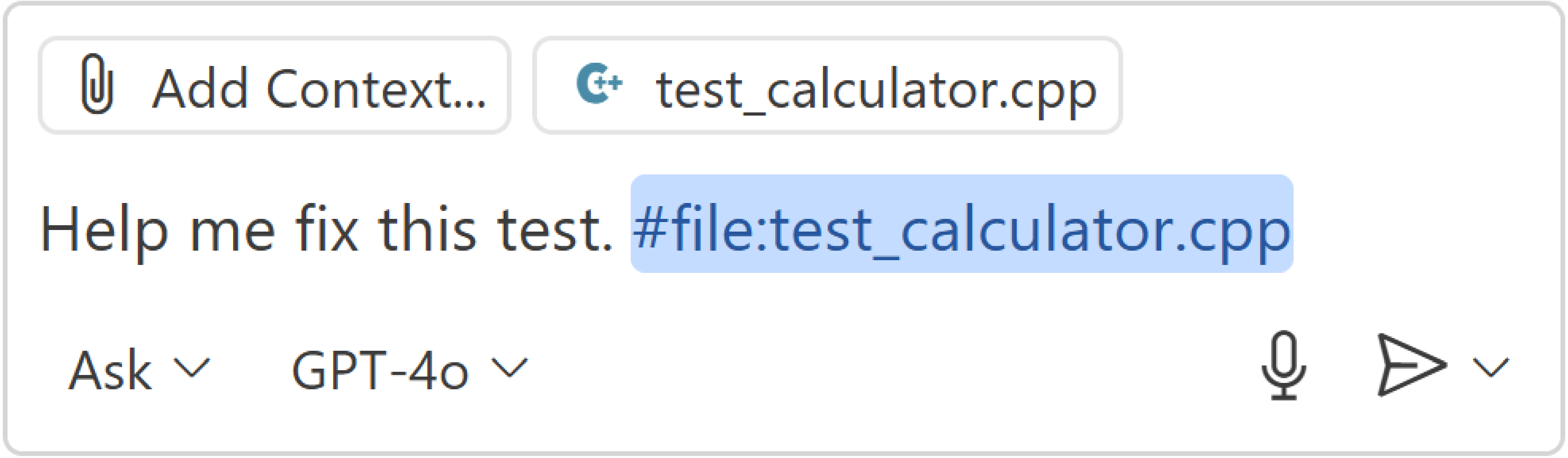


November 2022


Chat interface
LLM
Type “Help me fix this test.”
Paste unit test source code
Press send
Compute completion text
Display result

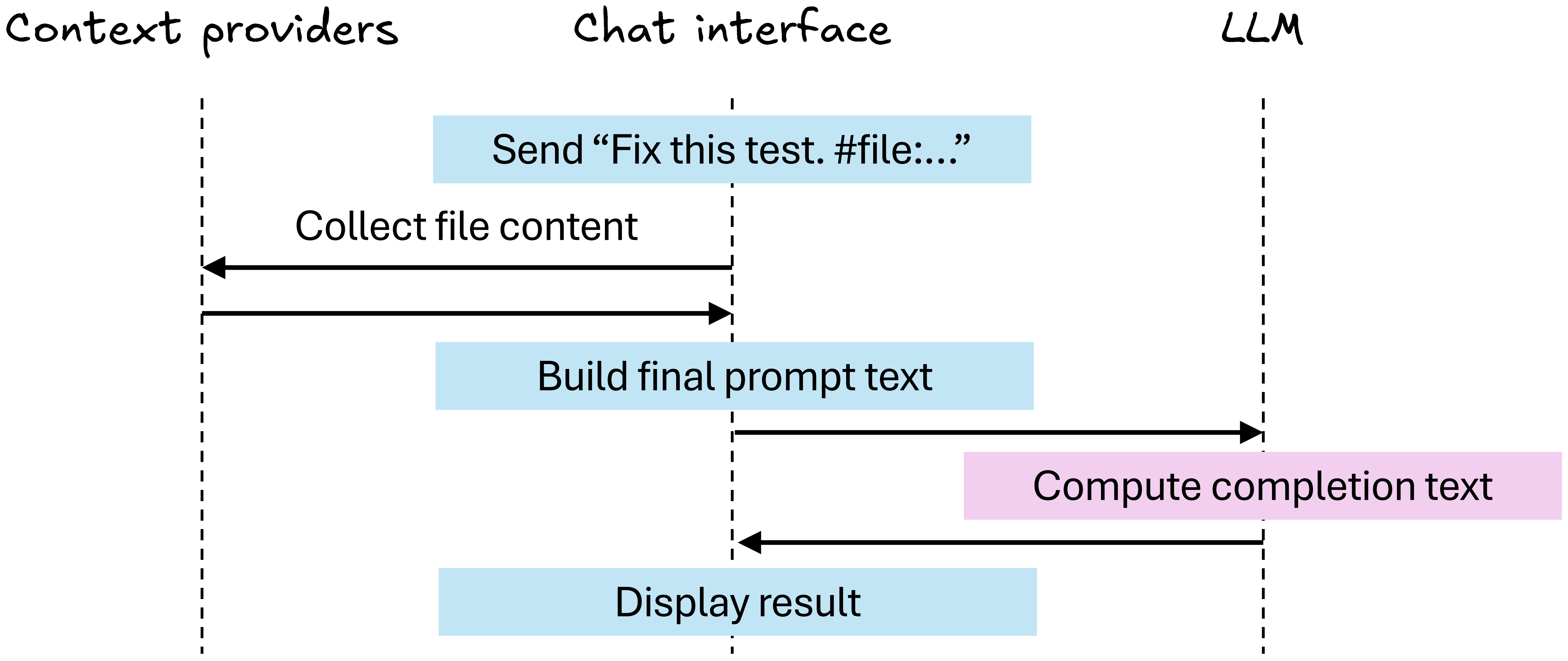

November 2022

IDE integration and better context



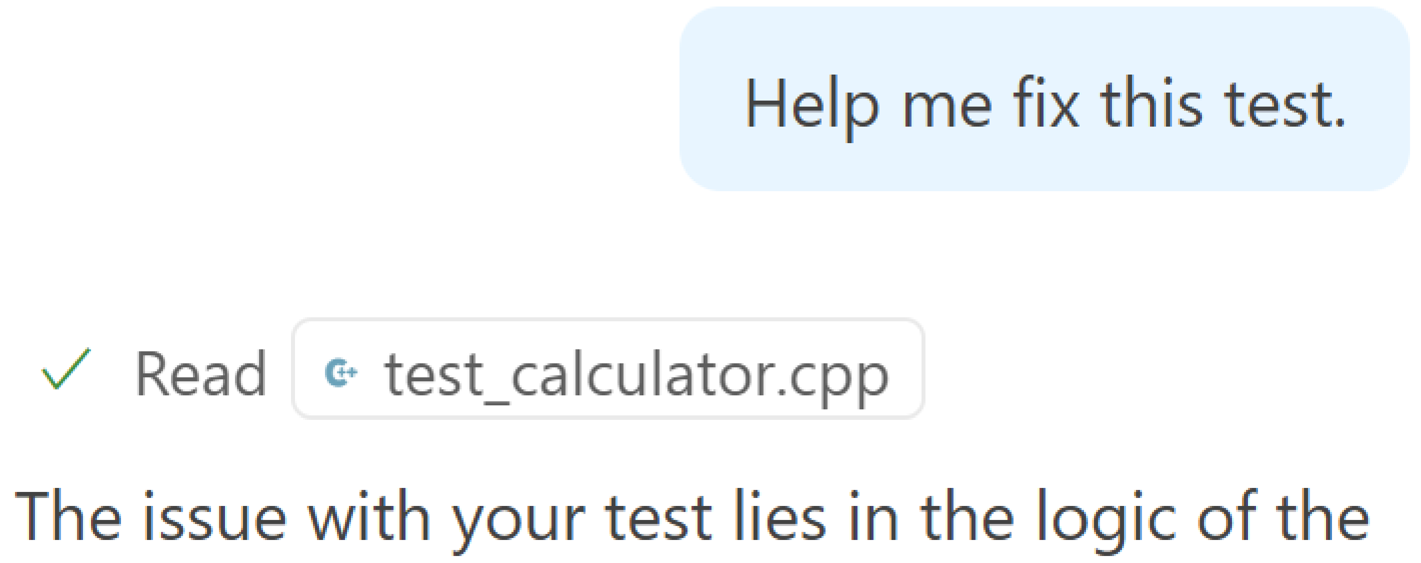
Help me fix this test. #file:test_calculator.cpp

Early 2023


Context providers
Chat interface
LLM
Send "Fix this test. #file:..."
Collect file content
Build final prompt text
Compute completion text
Display result

Early 2023

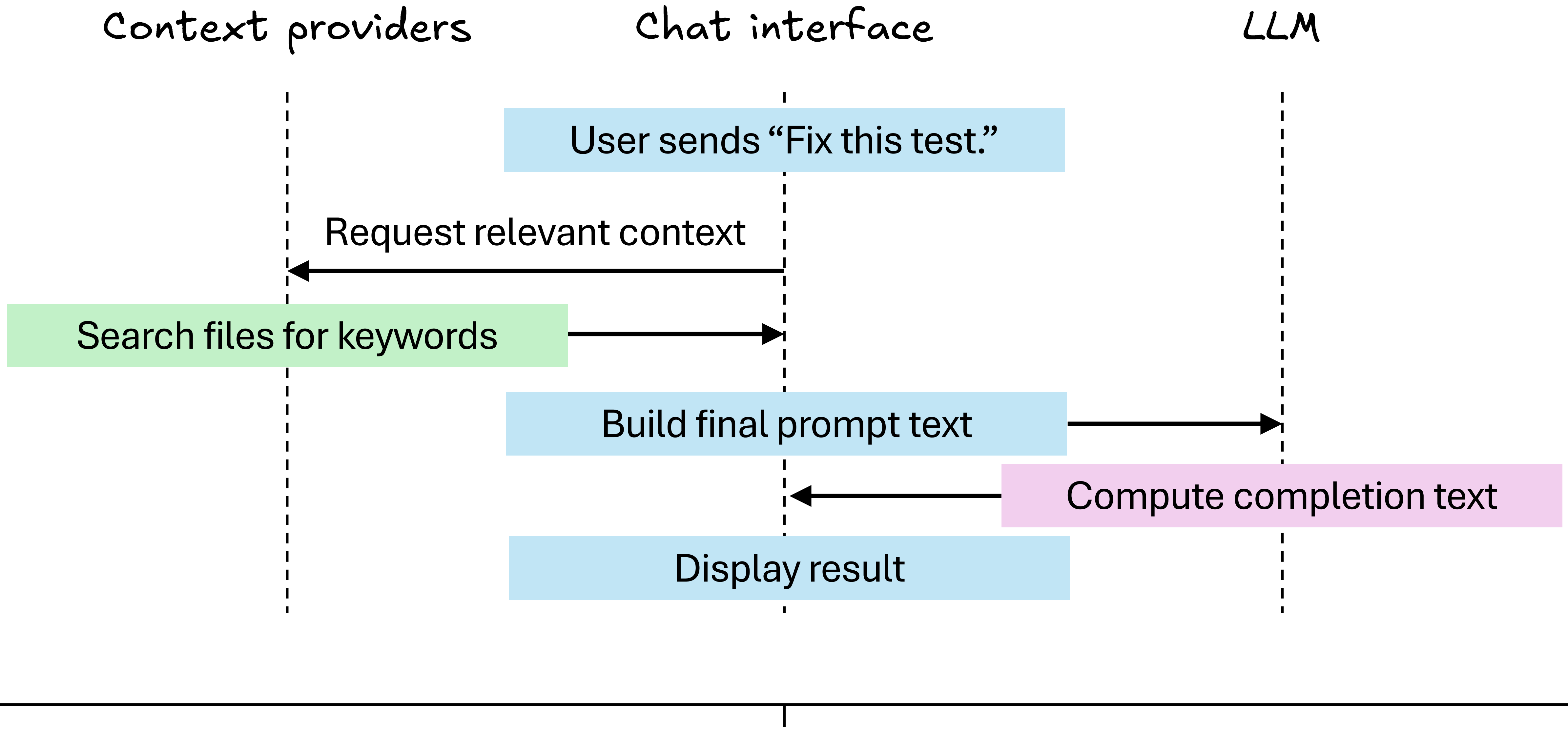
Automatic context (aka RAG)


Help me fix this test.

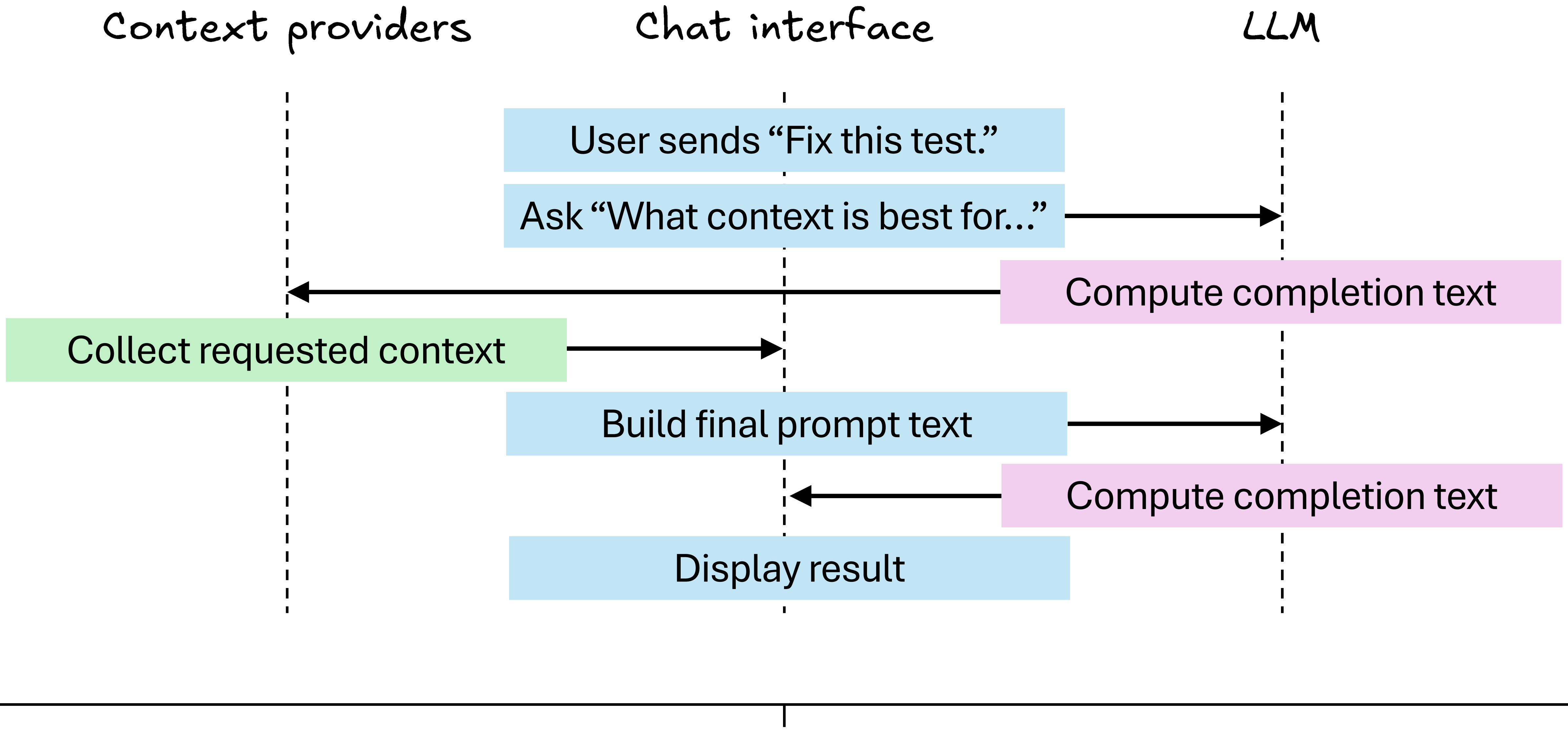
✓ Read test_calculator.cpp

The issue with your test lies in the logic of the

Mid 2023

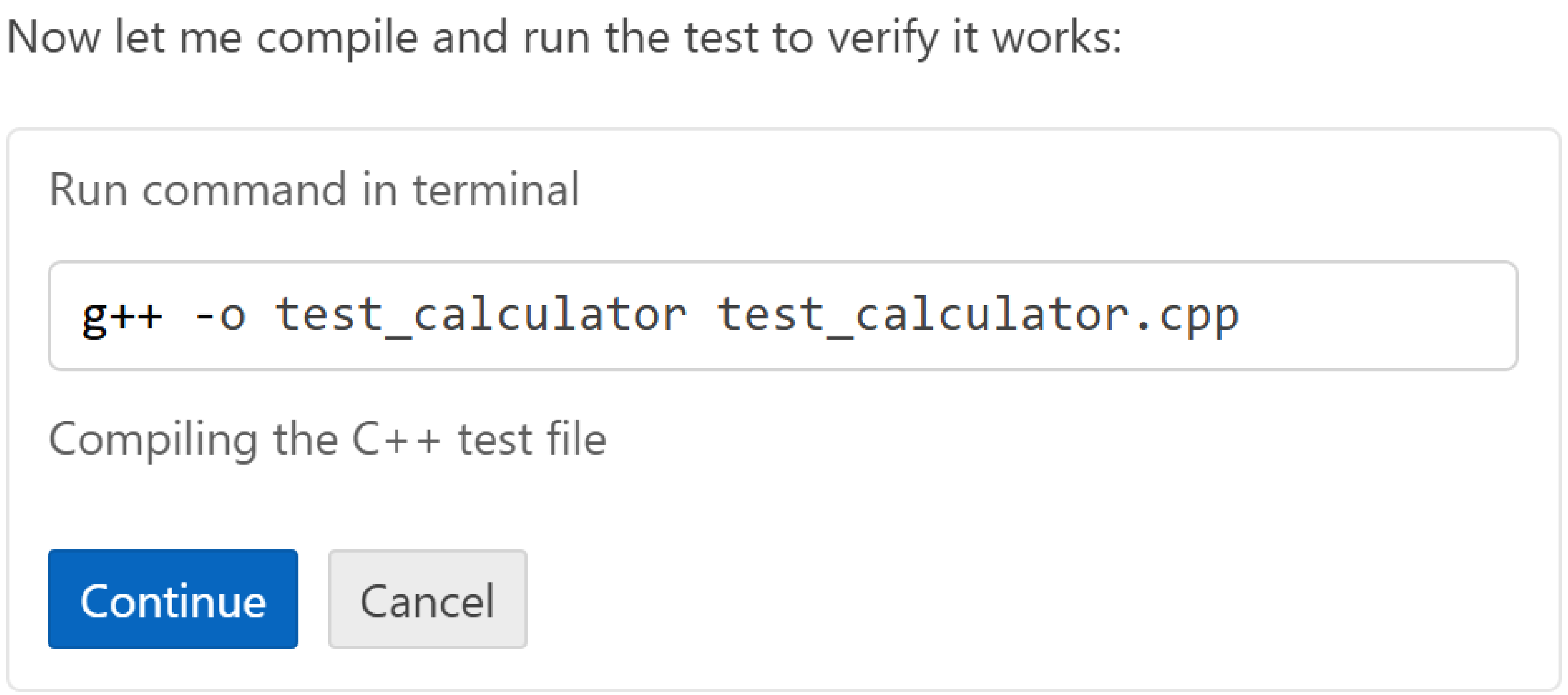

Context providers
Chat interface
LLM
User sends "Fix this test."
Request relevant context
Search files for keywords
Build final prompt text
Compute completion text
Display result

Mid 2023


Context providers
Chat interface
LLM
User sends “Fix this test.”
Ask “What context is best for…”
Compute completion text
Collect requested context
Build final prompt text
Compute completion text
Display result

Mid 2023

Agents and tool calling



Now let me compile and run the test to verify it works:

Run command in terminal

```
g++ -o test_calculator test_calculator.cpp
```

Compiling the C++ test file

Late 2024 – early 2025

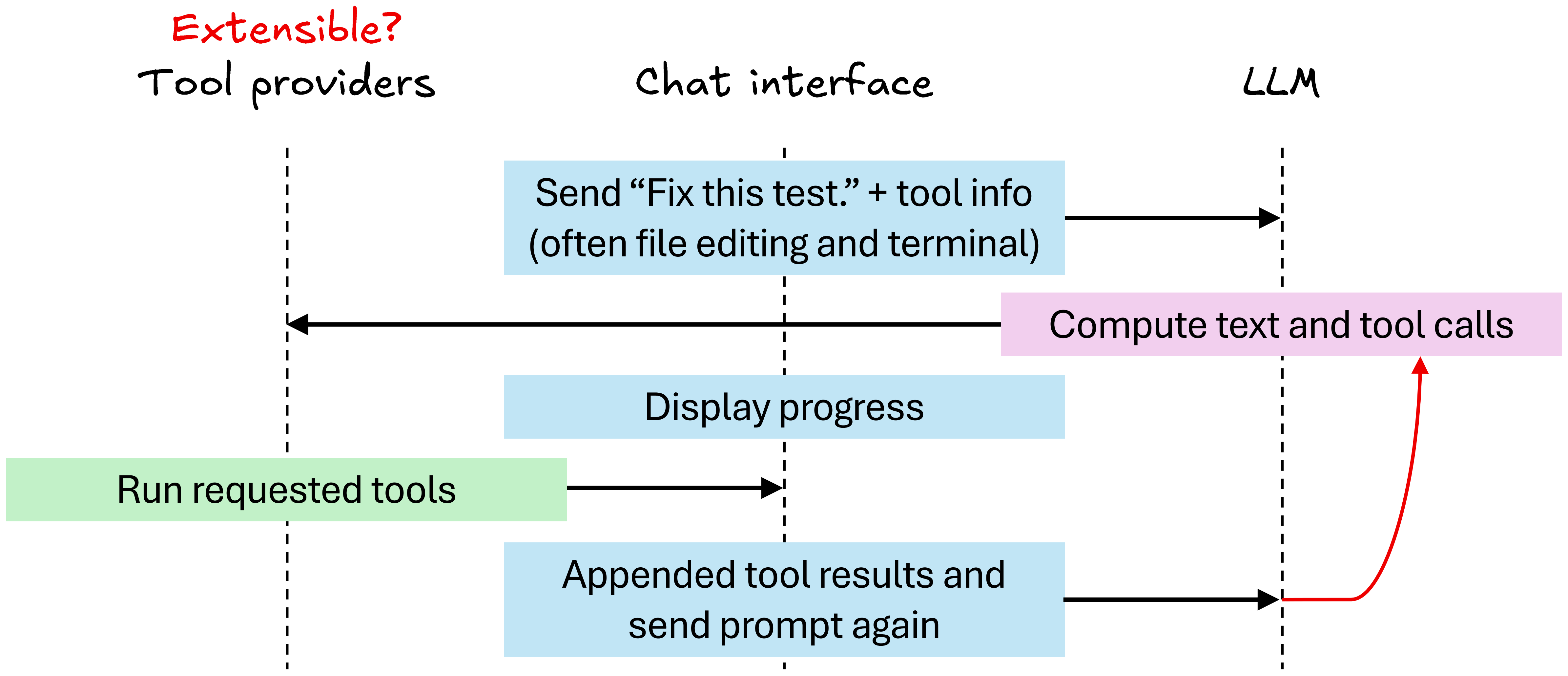
Tools provide **information** or take **actions**

```
{
    "name": "add",
    "description": "Computes the sum of two numbers",
    "parameters": {
        "type": "object",
        "properties": {
            "a": {
                "type": "number",
                "description": "The first number to add"
            },
            "b": {
                "type": "number",
                "description": "The second number to add"
            },
        },
        "required": ["a", "b"]
    }
}
```

Late 2024 – early 2025

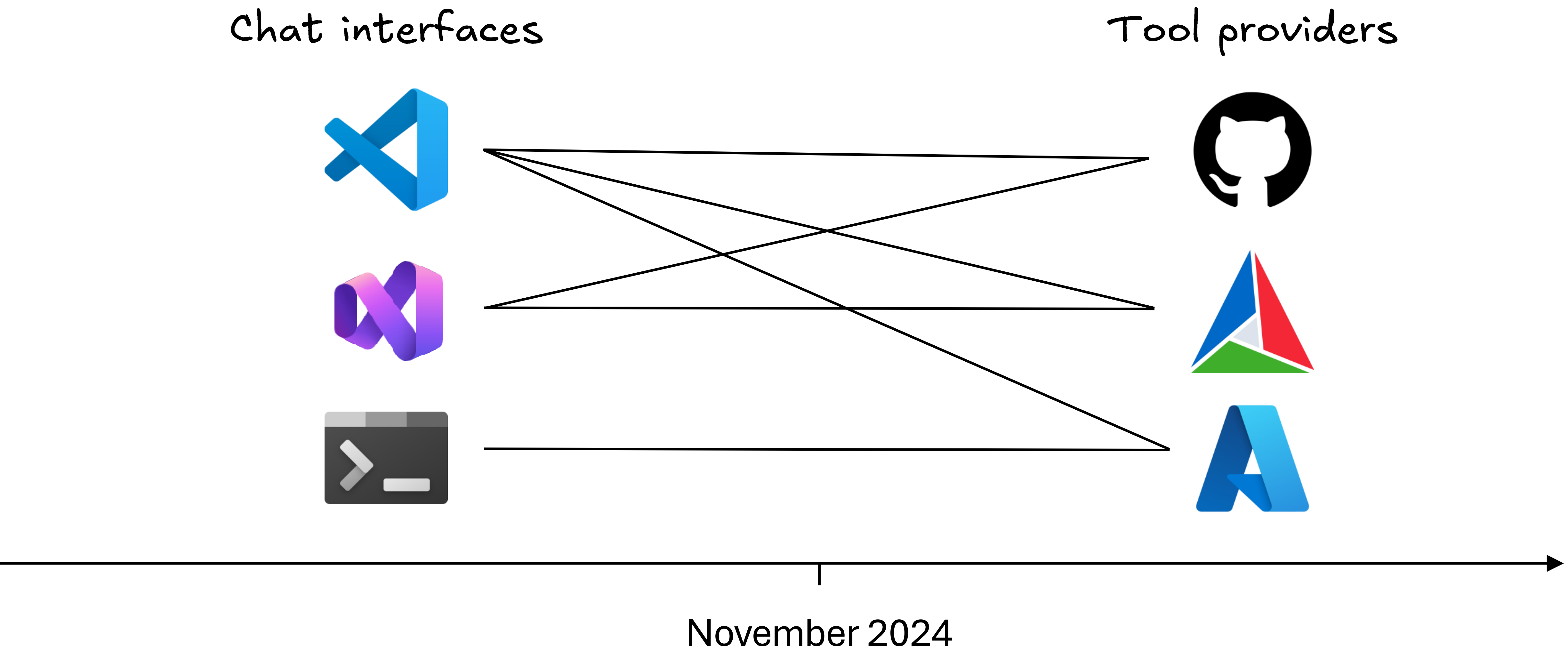
Client	Send `What is 37 + 94?` (plus tool metadata)
LLM	`37 + 94 is` **`<tool>add(37,94)</tool>`**
Client	Locally run add() function. Result is 131.
Client	Send `37 + 94 is <tool>add(37,94)</tool>` **`<tool_response>131</tool_response>`**
LLM	**`131`**
Client	Render response as “37 + 94 is 131”

Late 2024 – early 2025

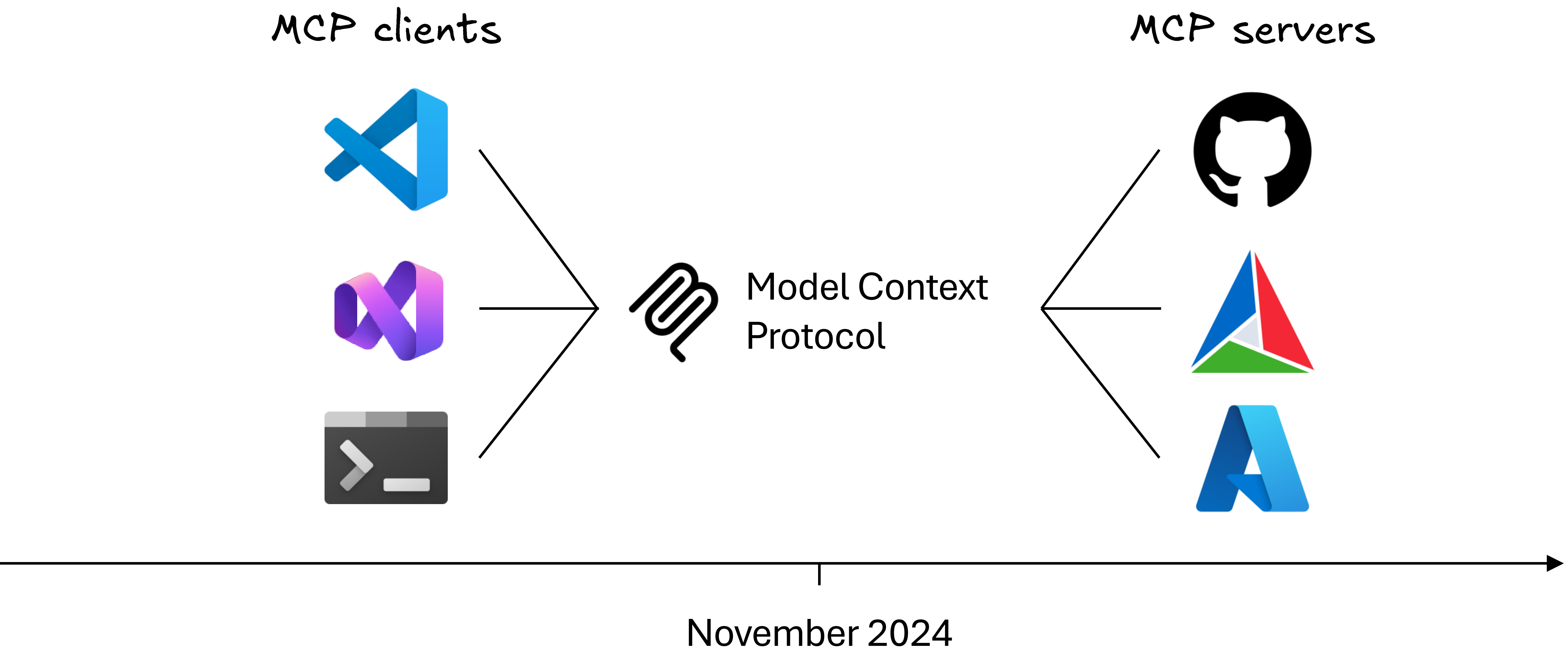

Extensible?
Tool providers
Chat interface
LLM
Send “Fix this test.” + tool info (often file editing and terminal)
Compute text and tool calls
Display progress
Run requested tools
Appended tool results and send prompt again

Late 2024 – early 2025

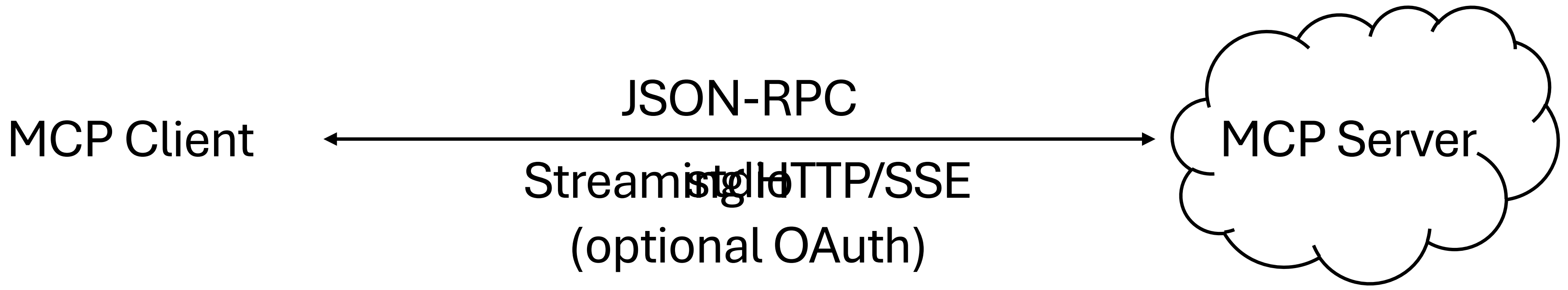
Interoperable tools



Interoperable tools



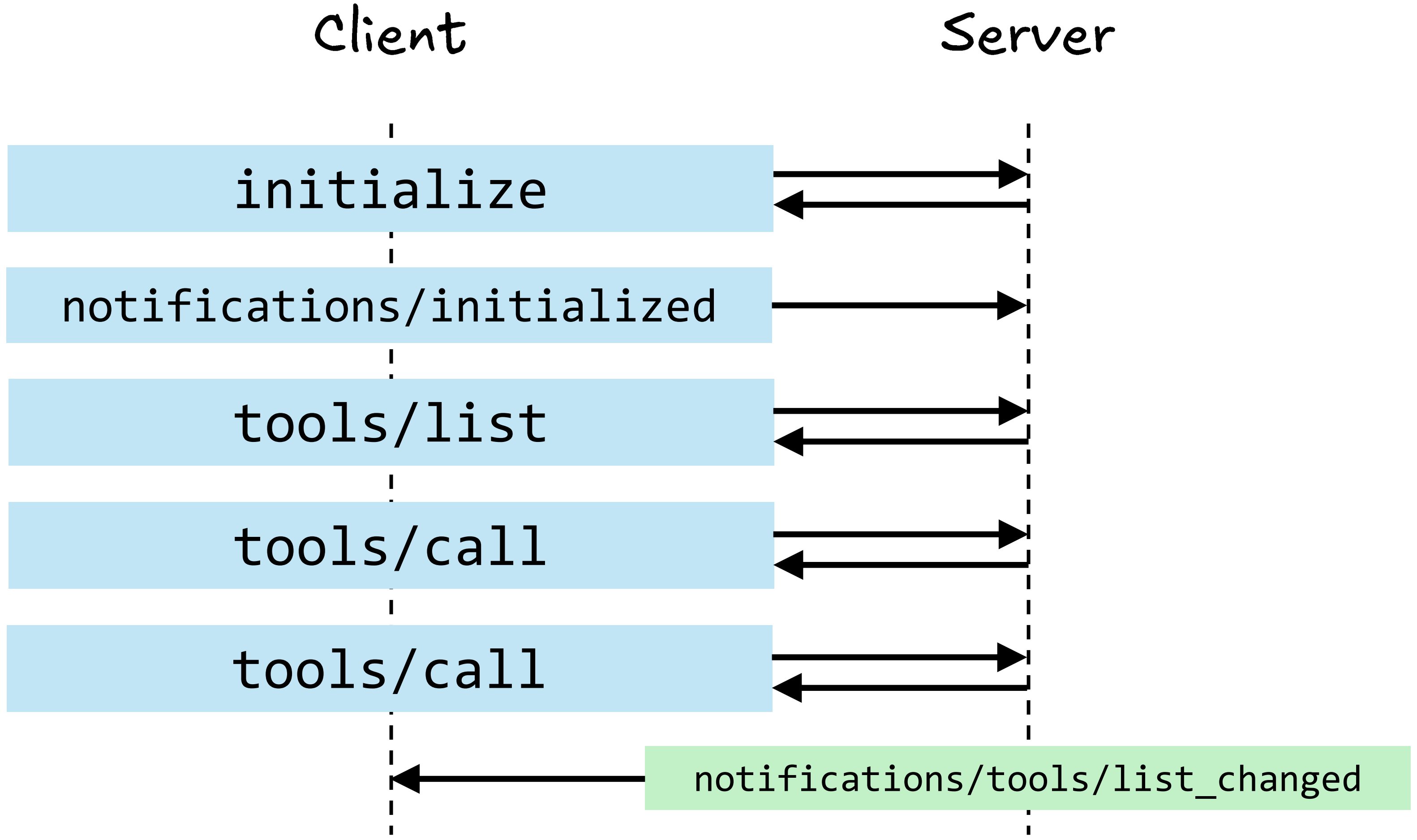
MCP under the hood



MCP under the hood

initialize
notifications/initialized
tools/list
tools/call

prompts/...
resources/...
sampling/createMessage
elicitation/create
completion/complete
logging/setLevel
notifications/...
roots/list
ping



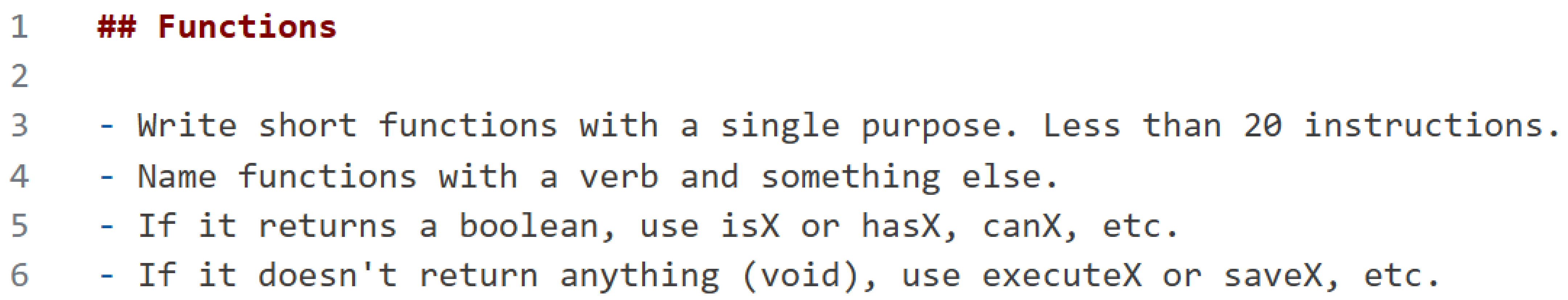
modelcontextprotocol.io/specification/2025-06-18/schema

Demo: write an MCP server

Best practices for MCP

You might not need MCP if…

…you want to add a small amount of information to every request.



```markdown
## Functions

- Write short functions with a single purpose. Less than 20 instructions.
- Name functions with a verb and something else.
- If it returns a boolean, use isX or hasX, canX, etc.
- If it doesn't return anything (void), use executeX or saveX, etc.
```

Custom instructions/rules and prompt files are commonly available

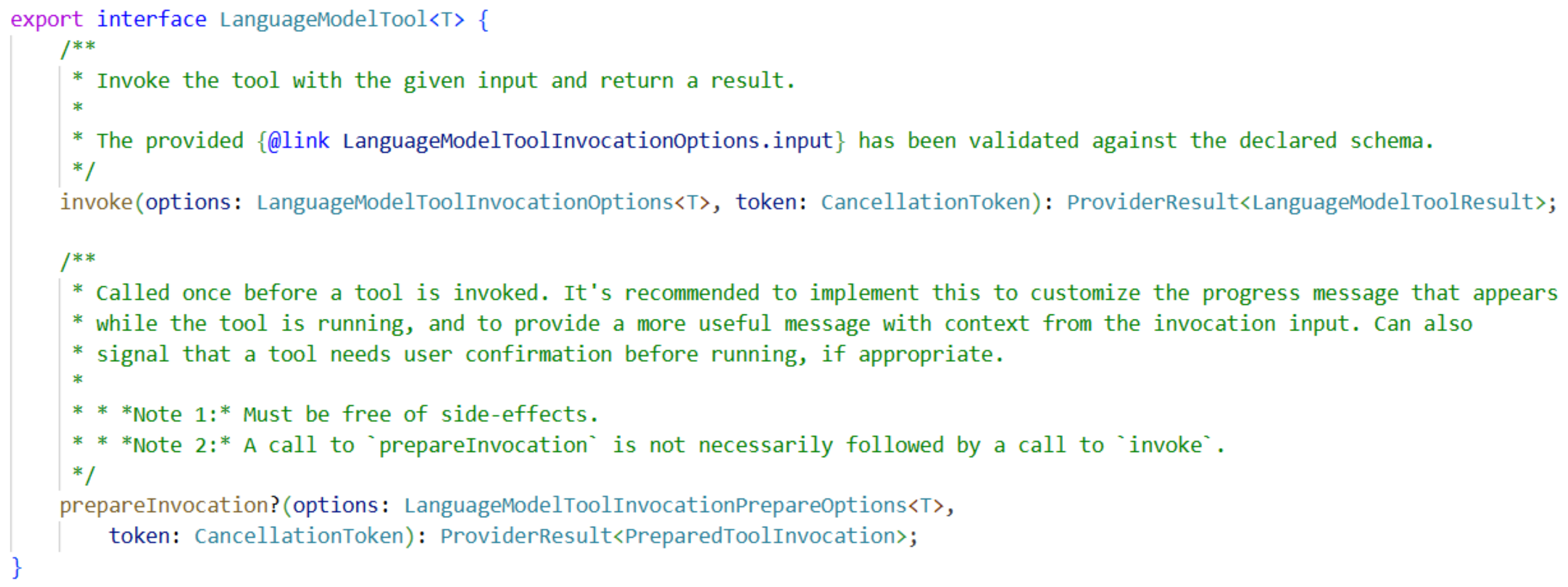
code.visualstudio.com/docs/copilot/customization/overview

You might not need MCP if…

…you can leverage built-in tools like run_in_terminal or web search.

You might not need MCP if…

…you only need to support one IDE or chat interface.



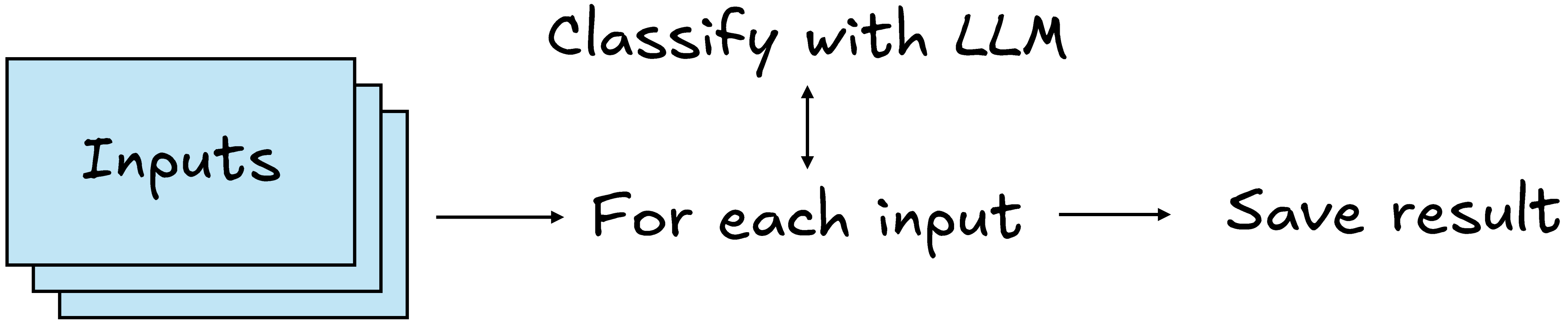
```
export interface LanguageModelTool<T> {
    /**
     * Invoke the tool with the given input and return a result.
     *
     * The provided {@link LanguageModelToolInvocationOptions.input} has been validated against the declared schema.
     */
    invoke(options: LanguageModelToolInvocationOptions<T>, token: CancellationToken): ProviderResult<LanguageModelToolResult>;

    /**
     * Called once before a tool is invoked. It's recommended to implement this to customize the progress message that appears
     * while the tool is running, and to provide a more useful message with context from the invocation input. Can also
     * signal that a tool needs user confirmation before running, if appropriate.
     *
     * * *Note 1:* Must be free of side-effects.
     * * *Note 2:* A call to `prepareInvocation` is not necessarily followed by a call to `invoke`.
     */
    prepareInvocation?(options: LanguageModelToolInvocationPrepareOptions<T>,
        token: CancellationToken): ProviderResult<PreparedToolInvocation>;
}
```

Platform-specific APIs to provide tools are often richer and easier

code.visualstudio.com/api/extension-guides/ai/tools

You might not need MCP if...

...you have a highly structured workflow.



Better to write a normal program and call the LLM as needed.

Security and the "lethal trifecta"

Server **A**

Access to **Private Data**

Ability to **Externally Communicate**

Server **B**

Exposure to **Untrusted Content**

Server **C**

simonwillison.net/2025/Jun/16/the-lethal-trifecta

Security and the "lethal trifecta"

Private data:
Internal documents

CVE-2025-32711

Communication:
Pre-fetching image links

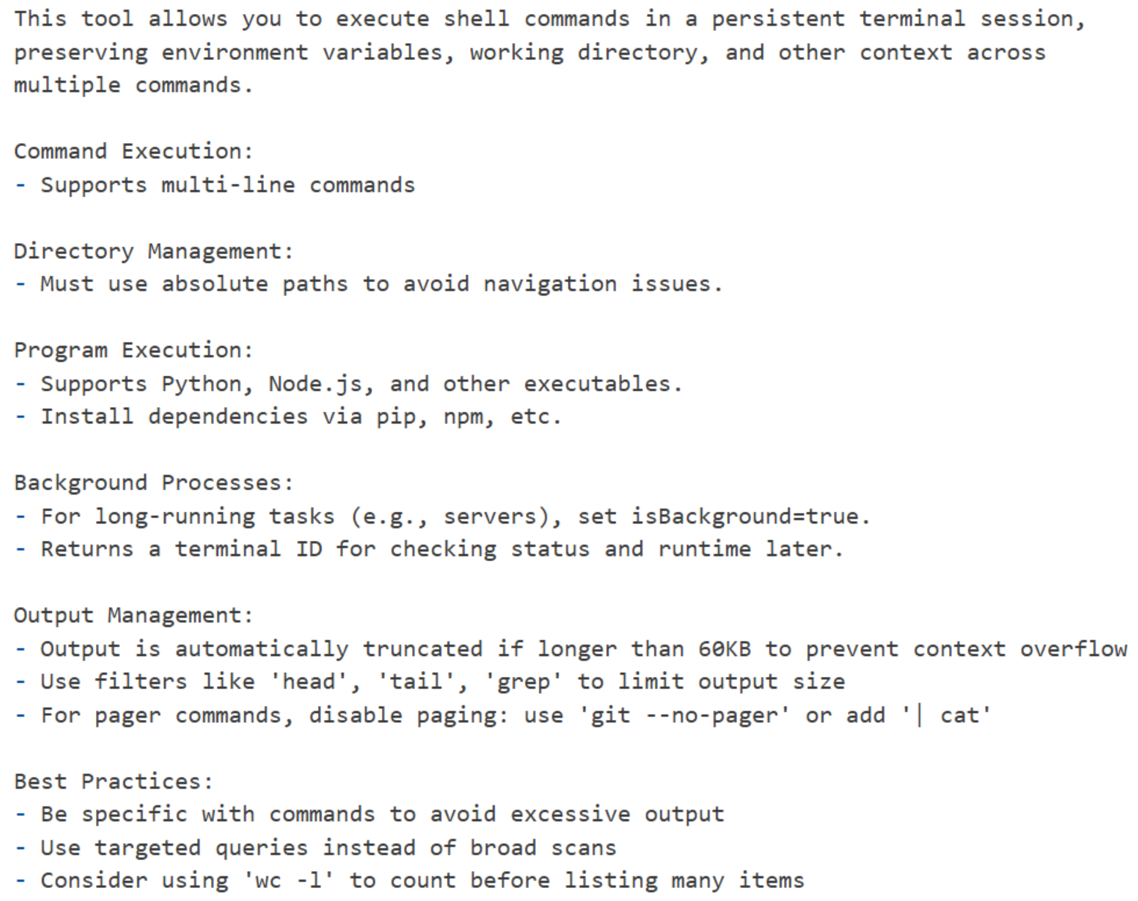
Untrusted content:
Email from attacker

Help the LLM use your tools

Write **great** tool descriptions

- What does it do?
- What information does it return?
- When should and shouldn't it be used?
- Any important limitations or constraints?

The LLM itself can help write these!



```
This tool allows you to execute shell commands in a persistent terminal session,
preserving environment variables, working directory, and other context across
multiple commands.

Command Execution:
- Supports multi-line commands

Directory Management:
- Must use absolute paths to avoid navigation issues.

Program Execution:
- Supports Python, Node.js, and other executables.
- Install dependencies via pip, npm, etc.

Background Processes:
- For long-running tasks (e.g., servers), set isBackground=true.
- Returns a terminal ID for checking status and runtime later.

Output Management:
- Output is automatically truncated if longer than 60KB to prevent context overflow
- Use filters like 'head', 'tail', 'grep' to limit output size
- For pager commands, disable paging: use 'git --no-pager' or add '| cat'

Best Practices:
- Be specific with commands to avoid excessive output
- Use targeted queries instead of broad scans
- Consider using 'wc -l' to count before listing many items
```

github.com/microsoft/vscode-copilot/blob/main/package.json

Help the LLM use your tools

Robustly handle invalid input and try LLM-friendly schema changes

```json
{
    "file": "/src/app.cpp",
    "operation": "definition",
    "offset": 5294
}
```

```json
{
    "file": "/src/app.cpp",
    "operation": "definition",
    "symbol": "App::run",
    "context": "\tif (ready) {\n\t\tApp::run();\n\t}\n"
}
```

Advanced MCP

Try **elicitation** and **sampling** messages for advanced tools

Elicitation: Server requests more information from the user *during tool invocation*

Sampling: Server sends messages directly to the LLM, proxied through the client

developer.microsoft.com/blog/can-you-build-agent2agent-communication-on-mcp-yes

Take our survey to win LEGO™ prizes!

aka.ms/cppcon/mcp

Microsoft at CppCon

Today

16:45	**Welcome to v1.0 of the meta::[[verse]]!** by Inbal Levi

Tomorrow

14:00	**MSVC C++ Dynamic Debugging: How We Enabled Full Debuggability of Optimized Code** by Eric Brumer
16:45	**It's Dangerous to Go Alone: A Game Developer Tutorial** by Michael Price

Friday

9:00	**Reflection-based JSON in C++ at Gigabytes per Second** by Daniel Lemire & Francisco Geiman Thiesen
13:30	**Duck-Tape Chronicles: Rust/C++ Interop** by Victor Ciura