

AI VIET NAM – AI COURSE 2025

Tutorial: Model Context Protocol

Trần Minh Nam

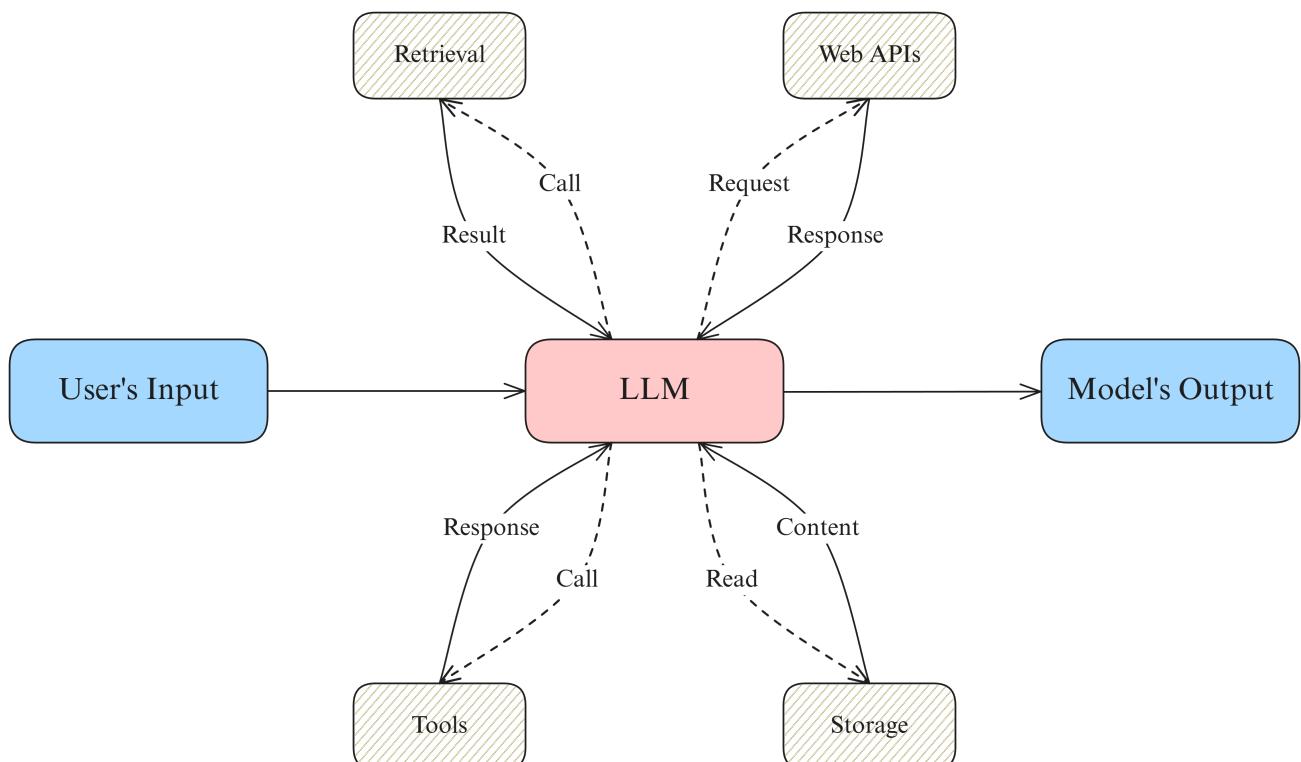
Nguyễn Quốc Thái

Đinh Quang Vinh

I. Giới thiệu

Khi các ứng dụng AI assistant ngày càng được sử dụng rộng rãi, rất nhiều nghiên cứu đã đầu tư vào việc nâng cao năng lực của mô hình, từ đó đạt được những bước tiến lớn về khả năng suy luận (reasoning ability) và hiệu suất của trên các benchmarks thực tế. Tuy nhiên, ngay cả những mô hình tiên tiến nhất vẫn bị hạn chế bởi sự tách biệt khỏi dữ liệu: mô hình không được tiếp tục huấn luyện để cập nhật những thông tin mới nhất. Ví dụ như GPT-4o có giới hạn dữ liệu huấn luyện đến tháng 6 năm 2024 hay của mô hình Llama 4 là tháng 8 năm 2024.

Để giải quyết vấn đề này, chúng ta tích hợp thông tin mới nhất vào mô hình bằng cách tìm kiếm thông tin liên quan đến vấn đề cần giải quyết từ các nguồn dữ liệu bên ngoài như cơ sở dữ liệu, kho tri thức, internet, v.v., và tích hợp những thông tin đó vào trong prompt của chúng ta trước khi gửi request đến mô hình (ví dụ như hệ thống RAG). Điều này cho phép mô hình có thể gián tiếp truy cập vào những thông tin không chứa trong knowledge.



Hình 1: LLM tích hợp tri thức ngoài

Tuy nhiên, mỗi công cụ hoặc cơ sở dữ liệu đều có cách thức truy cập và tương tác khác biệt, từ việc cấu trúc API, truy vấn SQL, đến các giao thức kết nối khác nhau. Điều này dẫn đến một thách thức lớn trong việc xây dựng các ứng dụng AI có thể kết nối với nhiều nguồn dữ liệu và công cụ khác nhau một cách hiệu quả, đó là việc dữ liệu bị phân mảnh (data fragmentation). Mỗi mô hình AI cần phải được tích hợp riêng với từng nguồn dữ liệu, dẫn đến việc开发者 phải viết code lặp đi lặp lại và khó khăn trong việc bảo trì cũng như thêm các tools khác vào mô hình.

Để giải quyết vấn đề này, **Model Context Protocol (MCP)** được phát triển như một giao thức thống nhất. MCP chuẩn hóa các kết nối giữa mô hình AI với các nguồn dữ liệu và công cụ khác nhau, giúp giảm thiểu sự phức tạp trong việc tích hợp và cho phép các mô hình AI truy cập vào dữ liệu một cách hiệu quả hơn.

Trong bài tutorial này, chúng ta sẽ tìm hiểu về Model Context Protocol (MCP), kiến trúc và các thành phần của MCP, và cuối cùng là xây dựng một công cụ tích hợp vào mô hình AI của chúng ta với MCP.

Mục lục

I.	Giới thiệu	1
II.	Model Context Protocol	4
II.1.	Tổng quan	4
II.2.	Kiến trúc của MCP	13
II.3.	MCP Client Features	17
II.4.	MCP Server Features	24
II.5.	Các cơ Chế Truyền Tải trong MCP	42
II.6.	Chạy Library MCP Server và thiết kế MCP Client để kiểm tra	49
III.	Cài đặt Personal Database MCP Server	57
III.1.	Tạo thư mục project và cài thư viện	57
III.2.	Chuẩn bị dữ liệu và văn bản	58
III.3.	Xây dựng Vector Store	60
III.4.	Xây dựng Retriever	65
III.5.	Xây dựng MCP Server	69
III.6.	Chạy MCP Server và tích hợp vào Claude Desktop	81
III.7.	Sử dụng Prompts, Resources và Tools trên Claude Desktop	84
IV.	Câu hỏi trắc nghiệm	88
IV.1.	Câu hỏi	88
IV.2.	Đáp án và giải thích	89
V.	Tài liệu tham khảo	91
	Phụ lục	94

II. Model Context Protocol

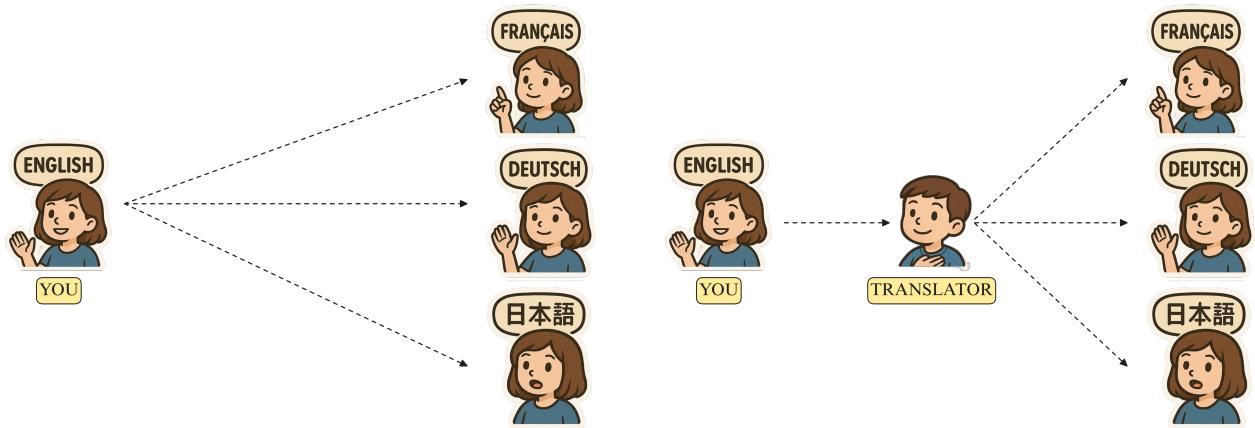
II.1. Tổng quan

II.1.1. Model Context Protocol là gì?

Model Context Protocol (MCP) là một giao thức mở, giúp chuẩn hóa kết nối giữa các ứng dụng cung cấp dữ liệu và các mô hình ngôn ngữ lớn. Chúng ta có thể hình dung MCP giống như một cổng USB-C dành riêng cho các ứng dụng AI. Nếu cổng USB-C giúp chúng ta dễ dàng kết nối thiết bị điện tử của chúng ta (như laptop, điện thoại, v.v) với nhiều phụ kiện khác nhau (điện thoại di động, màn hình, action camera, v.v.), thì MCP cũng giúp chúng ta kết nối các mô hình ngôn ngữ lớn với những nguồn dữ liệu (databases) và công cụ (tools) theo một quy chuẩn được định sẵn.

Để minh họa cho vai trò của Model Context Protocol (MCP), ta có thể hình dung một tình huống thực tế như sau:

- Giả sử bạn chỉ biết tiếng Anh, và trong một buổi tiệc giao lưu, bạn gặp gỡ ba người bạn quốc tế đến từ các nước khác nhau. Một người chỉ nói được tiếng Pháp, một người chỉ nói tiếng Đức, và người còn lại chỉ nói tiếng Trung Quốc (Hình 2a).
- Bạn rất muốn kết bạn và trao đổi thông tin với từng người, nhưng rào cản ngôn ngữ khiến việc này trở nên cực kỳ khó khăn. Nếu bạn quyết định tự mình học từng ngôn ngữ một, bạn sẽ phải đầu tư rất nhiều thời gian, công sức và có thể không hiệu quả ngay tức thì. Ngược lại, nếu từng người bạn mới cố gắng học tiếng Việt, thì thử thách cũng không hề nhỏ.
- Tuy nhiên, giả sử trong buổi tiệc đó xuất hiện thêm một người phiên dịch thành thạo cả bốn ngôn ngữ: tiếng Việt, tiếng Anh, tiếng Đức và tiếng Pháp. Người phiên dịch này trở thành trung gian giúp bạn giao tiếp dễ dàng với từng người bạn mới mà không cần trực tiếp học thêm ngôn ngữ nào.
- Cuộc trò chuyện diễn ra vô cùng suôn sẻ nhờ sự hỗ trợ của người phiên dịch, mọi hiểu lầm hoặc khó khăn đều được giải quyết nhanh chóng (Hình 2b). Bạn cảm thấy thoải mái hơn, tiết kiệm thời gian và công sức hơn.



(a) Việc kết nối với nhau sẽ khó khăn hơn khi gặp rào cản về ngôn ngữ (b) Việc kết nối sẽ trở nên dễ dàng hơn khi có người phiên dịch làm cầu nối

Hình 2: Ví dụ về MCP thông qua hình ảnh thông dịch viên

Trong bối cảnh xây dựng các ứng dụng AI, Model Context Protocol đóng vai trò tương tự như người phiên dịch. MCP hoạt động như một cầu nối trung gian, giúp các hệ thống, ứng dụng và dữ liệu “nói chuyện” và hiểu nhau một cách nhanh chóng và hiệu quả mà không cần phải mất thời gian thay đổi hoặc điều chỉnh từng thành phần riêng lẻ.

II.1.2. Tại sao MCP ra đời?

Để hiểu tại sao MCP ra đời, chúng ta sẽ đi tìm hiểu về function calling.

Function calling Function calling là một tính năng của các mô hình ngôn ngữ lớn (LLM), cho phép các mô hình không chỉ dừng lại ở việc tạo ra văn bản mà còn có thể tương tác với các công cụ và dịch vụ bên ngoài. Khi chúng ta hỏi một chatbot về thời tiết hôm nay, thay vì chỉ đưa ra câu trả lời chung chung (ví dụ như: “Tôi không thể trả lời câu hỏi của bạn vì tôi không có kiến thức về thời tiết ngày hôm nay”), mô hình có thể gửi request đến API thời tiết để lấy thông tin chính xác và được cập nhật theo thời gian thực.

Về bản chất, function calling giúp LLM có khả năng thực hiện các hành động cụ thể thông qua việc gọi các hàm được định nghĩa trước. Cơ chế này rất hữu ích khi chúng ta cần xây dựng các ứng dụng có sự tương tác phức tạp như chatbot hỗ trợ khách hàng có thể tra cứu đơn hàng, AI assistant có thể gửi email, hoặc hệ thống phân tích có thể truy vấn cơ sở dữ liệu.

Function calling chủ yếu được sử dụng trong 3 trường hợp chính như sau:

1. Mở rộng kiến thức: Mô hình có thể truy cập và khai thác dữ liệu từ các nguồn bên ngoài như cơ sở dữ liệu, API, kho kiến thức chuyên biệt, v.v., giúp cung cấp thông tin mới theo thời gian thực.
2. Nâng cao chức năng: Bằng cách gửi request đến các công cụ hỗ trợ như máy tính, thư viện đồ họa, biểu đồ, v.v., mô hình có thể thực hiện tính toán và khai thác dữ liệu dễ dàng hơn.
3. Thực hiện hành động: Mô hình có thể tương tác trực tiếp với hệ thống qua API để thực hiện các hành động như đặt lịch, tạo hóa đơn, gửi email, điều khiển thiết bị smarthome,

v.v. Tất cả đều được thực hiện tự động và chính xác dựa theo lệnh của mô hình.

Cách function calling hoạt động Để ví dụ về việc function calling hỗ trợ nâng cao chức năng của LLM và đồng thời hiểu được quy trình hoạt động của function calling, chúng ta hãy so sánh việc sử dụng LLM có function calling và không có function calling để tính toán giá trị của biểu thức toán học sau đây:

$$9.896 - 4.012 + (-13.23456908) - (-2^{-1.05}) \quad (1)$$

Trước khi tích hợp function calling, ta sử dụng mô hình Gemini 2.0 Flash để tính toán biểu thức trên:

Dùng LLM giải quyết bài toán trên khi không dùng thêm tool ngoài

```

1 from google import genai
2 from google.genai import types
3
4 client = genai.Client(api_key="your_api_key")
5 config = types.GenerateContentConfig(temperature=0.0, top_k=1, max_output_tokens=10)
6
7 response = client.models.generate_content(
8     model="gemini-2.0-flash",
9     contents="Just give me the answer directly without calling any external tools or
support: 9.896 - 4.012 + (-13.23456908) -
(- 2**(- 1.05)) = ?",
10    config=config,
11 )
12
13 print(response.text)

```

và nhận được kết quả như sau:

Kết quả khi dùng LLM mà không có external tool để giải bài toán trên

-7.17

Tuy nhiên, nếu sử dụng máy tính, kết quả chính xác là -6.867600915537576 , khác với kết quả mô hình trả về là -6.86828308 .

Bây giờ, chúng ta sẽ tích hợp function calling vào LLM để tính toán biểu thức trên theo các bước như sau:

- Ta sẽ định nghĩa và miêu tả hàm `evaluate_math_expression` để hàm nhận vào một biểu thức dưới dạng string như sau:

Định nghĩa và miêu tả hàm `evaluate_math_expression`

```

1 # Function Declaration
2 def evaluate_math_expression(expression: str, precision: int = None) -> float:
3     allowed_names = {k: v for k, v in math.__dict__.items() if not k.startswith(
4         "__")}
        allowed_names.update({"abs": abs, "round": round, "min": min, "max": max, "pow"

```

```

                                ": pow)

5
6     try:
7         result = eval(expression, {"__builtins__": {}}, allowed_names)
8         if precision is not None:
9             return round(result, precision)
10        return result
11    except Exception as e:
12        raise ValueError(f"Error evaluating expression: {e}")
13
14 # Function Definition
15 evaluate_math_expression_function = {
16     "name": "evaluate_math_expression",
17     "description": "Evaluates a mathematical expression given as a string and
18     returns the numeric result.",
19     "parameters": {
20         "type": "object",
21         "properties": {
22             "expression": {
23                 "type": "string",
24                 "description": "The mathematical expression to evaluate, e.g., '3
25                 * (4 + 5) / 2'.",
26             },
27             "precision": {
28                 "type": "number",
29                 "description": "Number of decimal places to round the result to.",
30             },
31         },
32     }
}

```

Lưu ý rằng định nghĩa hàm (Function Definition) được đưa vào cùng với request gửi đến LLM. Còn việc khai báo hàm (Function Declaration) là để thực thi hàm khi hàm được gọi đến.

- Chúng ta sẽ gửi request đến LLM cùng với dữ liệu về hàm đã định nghĩa trước đó:

Request đến LLM

```

1 tools = types.Tool(function_declarations=[evaluate_math_expression_function])
2 config = types.GenerateContentConfig(tools=[tools], temperature=0.0, top_k=1)
3
4 conversation = [
5     types.Content(
6         role="user", parts=[types.Part(text="9.896 - 4.012 + (-13.23456908) - (- 2
7             **(- 1.05))?"")]
8     )
9 ]
10 response = client.models.generate_content(
11     model="gemini-2.0-flash",

```

```

12     contents=conversation,
13     config=config,
14 )
15 print(response.candidates[0].content.parts[0].function_call)

```

2. Khi mô hình quyết định gọi hàm, nó sẽ trả về tên hàm cần gọi cùng các đối số (input arguments) dưới dạng dữ liệu có cấu trúc:

Kết quả trả về với lệnh function calling

FunctionCall(id=None, args='expression': '9.896 - 4.012 + (-13.23456908) - (- 2**(-1.05))', name='evaluate_math_expression')

3. Chúng ta tiến hành phân tích phản hồi từ mô hình để trích xuất hàm được chọn và xử lý việc gọi hàm tương ứng, cụ thể ở đây là hàm `evaluate_math_expression`.

Thực thi hàm evaluate_math_expression với arguments được trả về

```

1 tool_call = response.candidates[0].content.parts[0].function_call
2
3 if tool_call.name == "evaluate_math_expression":
4     result = evaluate_math_expression(**tool_call.args)
5     print(f"Function execution result: {result}")

```

và output nhận được như sau:

Kết quả thực thi hàm

Function execution result: -6.867600915537576

4. Chúng ta cung cấp kết quả gọi hàm cho mô hình để mô hình có thể tích hợp chúng vào phản hồi cuối cùng của mình và gửi conversation đã update kết quả function call đến LLM như sau:

Gửi kết quả chạy hàm cho LLM trong luồng conversation

```

1 # Append function call and result of the function execution to the conversation
2 function_response_part = types.Part.from_function_response(
3     name=tool_call.name,
4     response={"result": result},
5 )
6 conversation.append(response.candidates[0].content)
7 conversation.append(types.Content(role="user", parts=[function_response_part]))
8
9 final_response = client.models.generate_content(
10    model="gemini-2.0-flash",
11    contents=conversation,
12    config=config,

```

```

13 )
14
15 print(final_response.text)

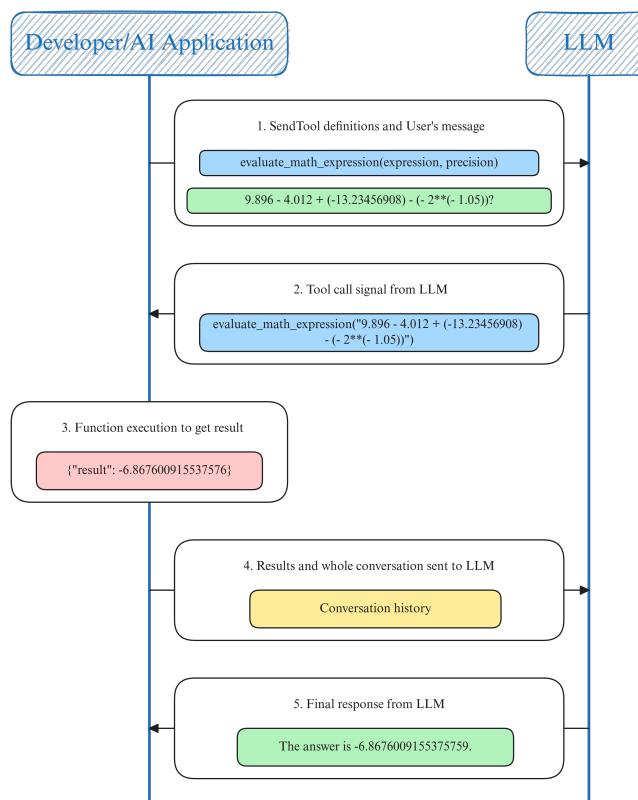
```

5. Cuối cùng, chúng ta nhận được kết quả như sau:

Kết quả trả về

The answer is -6.867600915537576.

Như vậy, thông qua function calling, chúng ta đã tăng thêm khả năng tính toán cho LLM. Ngoài ra, chúng ta có thể tích hợp vô số hàm khác nhau như gửi email, kiểm tra thời tiết, truy vấn cơ sở dữ liệu. Từ đó, ta thấy function calling hỗ trợ LLM trong việc tương tác với các công cụ, giúp LLMs giải quyết được những vấn đề còn tồn đọng trong bản thân mô hình như không thể tính toán chính xác, khó khăn trong quá trình tương tác trực tiếp với môi trường, v.v.



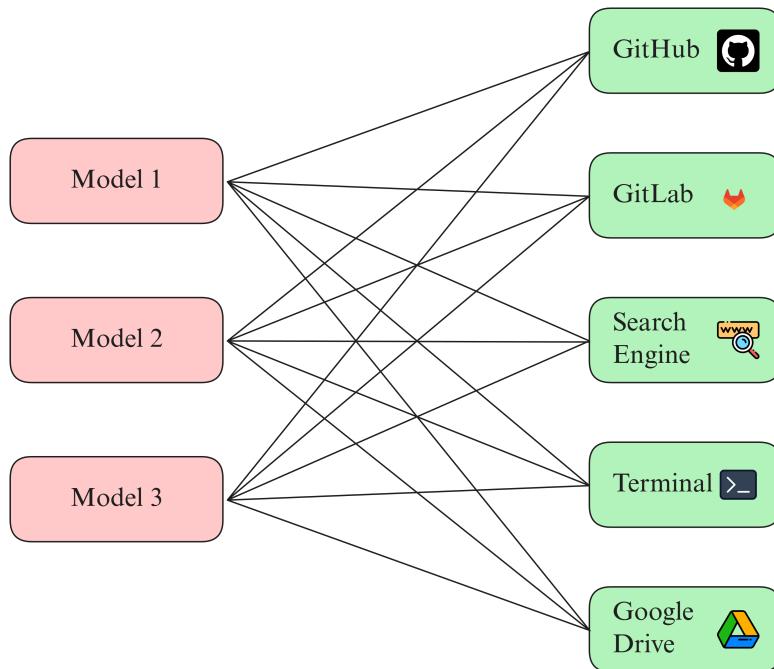
Hình 3: Function Calling Workflow

Quy trình làm việc của function calling ở trên có thể được minh họa từng bước như trong Hình 3. Chúng ta có thể thấy tools được implement và thực thi ở bên phía AI applications của chúng ta. Nếu chúng ta có sự thay đổi về mặt logic của tools, cấu trúc của tools hay số lượng tham số của tools, chúng ta phải viết lại tools cũng như định nghĩa của tools. Cuối cùng là khởi động lại ứng dụng AI.

Vấn đề của function calling và cách MCP giải quyết bài toán Function calling hỗ trợ LLM trong việc tăng khả năng xử lý thông tin một cách chính xác, giúp LLM có thể hành động và tương tác với môi trường, bổ sung kiến thức mới cho mô hình. Tuy nhiên, vấn đề nảy sinh khi ta cần tích hợp nhiều tools vào các ứng dụng AI của chúng ta.

Trước khi MCP ra đời, để kết nối một mô hình AI với nhiều nguồn dữ liệu (như GitHub, GitLab, databases, local file system, v.v.) hoặc công cụ (như web browser, Slack, Google Drive, v.v.), các nhà phát triển phải viết nhiều connector riêng biệt. Các công cụ pháp lúc đó thường mang tính thủ công như: implement logic riêng cho từng công cụ, design prompt để mô hình gọi riêng các tool, hoặc viết tool để gọi API đến từng nền tảng.

Hệ quả là một bài toán tích hợp nổi tiếng: bài toán $M \times N$. Nghĩa là, nếu có M mô hình AI và N nguồn dữ liệu hay công cụ khác nhau, thì chúng ta cần xây dựng đến $M \times N$ kết nối riêng lẻ, dẫn tới việc vừa phức tạp vừa khó mở rộng quá trình tích hợp các công cụ vào ứng dụng AI (minh họa trong Hình 4). Bên cạnh đó, các tools được implement và thực thi ở phía ứng dụng AI (xem Hình 3), dẫn tới việc phải khởi động lại ứng dụng AI khi có tools mới hoặc cập nhật các tools có sẵn.



Hình 4: Mỗi mô hình AI phải kết nối đến từng tools

Trong ví dụ về function calling ở phần trước, hàm `evaluate_math_expression` được định nghĩa với hai tham số `expression` và `precision`, và nếu tham số `precision` không được thiết lập, hàm sẽ trả về số chữ số thập phân một cách tùy ý. Tuy nhiên, ở một ứng dụng AI khác, hàm dùng để tính toán biểu thức có thể được định nghĩa dưới tên khác (ví dụ như `get_expression_value`, chỉ nhận một tham số đầu vào là biểu thức dưới dạng chuỗi, và trả về kết quả là giá trị được làm tròn đến hai chữ số thập phân dưới dạng chuỗi (xem đoạn code minh họa II.1.2.). Trong trường hợp hàm `get_expression_value` có sự thay đổi về mặt implementation để tích hợp thêm argument `precisions` (đại diện cho số chữ số thập phân sau dấu phẩy), developers phải sửa lại toàn bộ hàm, viết định nghĩa cho tool mới và khởi động lại ứng dụng AI. Những rắc rối trên bắt

nguồn từ việc chúng ta implement tools theo những cách khác nhau, dẫn đến sự bất đồng bô trong việc quản lí và sử dụng tools đó.

Một cách implement khác của hàm evaluate_math_expression

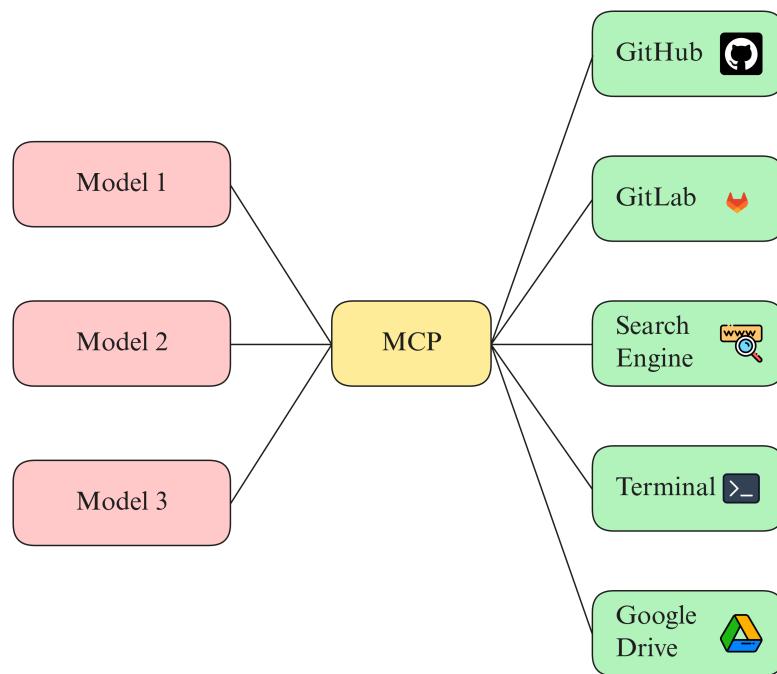
```

1 import ast
2 import operator as op
3
4 def get_expression_value(expression: str) -> str:
5
6     def _eval_node(node):
7         # Supported operators
8         OPERATORS = {
9             ast.Add: op.add,
10            ast.Sub: op.sub,
11            ast.Mult: op.mul,
12            ast.Div: op.truediv,
13            ast.Mod: op.mod,
14            ast.Pow: op.pow,
15            ast.USub: op.neg,
16            ast.UAdd: op.pos,
17            ast.FloorDiv: op.floordiv,
18        }
19        if isinstance(node, ast.Num): # for Python <3.8
20            return node.n
21        elif isinstance(node, ast.Constant): # for Python >=3.8
22            if isinstance(node.value, (int, float)):
23                return node.value
24            raise TypeError(f"Unsupported constant type: {type(node.value).__name__}")
25        elif isinstance(node, ast.BinOp):
26            left = _eval_node(node.left)
27            right = _eval_node(node.right)
28            operator = OPERATORS.get(type(node.op))
29            if operator is None:
30                raise TypeError(f"Unsupported binary operator: {type(node.op).__name__}")
31            return operator(left, right)
32        elif isinstance(node, ast.UnaryOp):
33            operand = _eval_node(node.operand)
34            operator = OPERATORS.get(type(node.op))
35            if operator is None:
36                raise TypeError(f"Unsupported unary operator: {type(node.op).__name__}")
37            return operator(operand)
38        else:
39            raise TypeError(f"Unsupported expression type: {type(node).__name__}")
40
41    node = ast.parse(expression, mode="eval")
42    result = _eval_node(node.body)
43    result = round(result, 2) # Round to two decimal places
44    return f"{result:.2f}"
45
46 print(get_expression_value("9.896 - 4.012 + (-13.23456908) - (- 2**(- 1.05)))")
47 ##### OUTPUT #####

```

48 ##### -6.87 #####

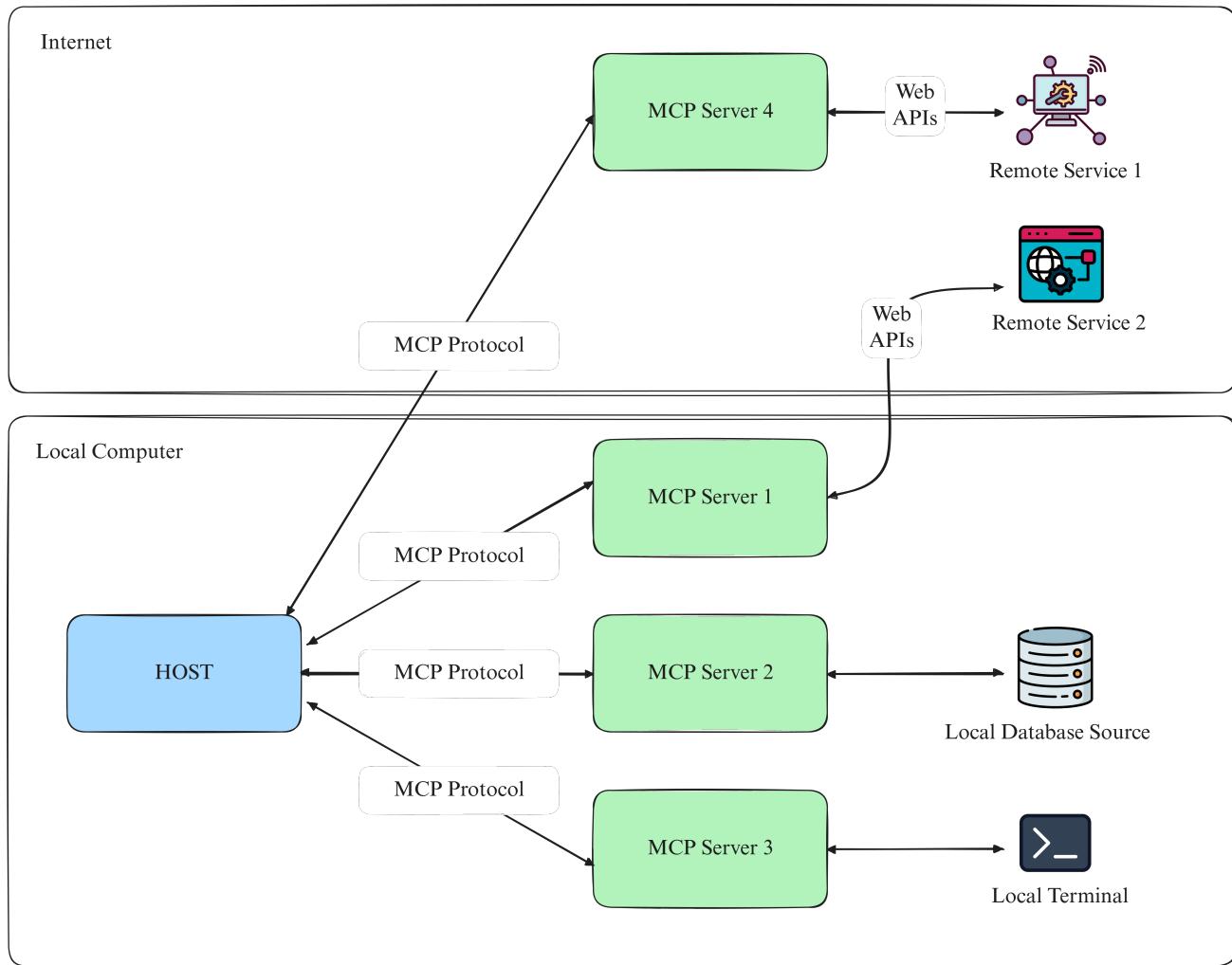
MCP giải quyết vấn đề này bằng cách đưa ra một giao thức chuẩn ở vị trí trung gian giữa LLM và các công cụ. Thay vì phải xây dựng $M \times N$ tích hợp riêng biệt giữa từng ứng dụng AI và từng công cụ, chúng ta chỉ cần $M + N$ lần triển khai: mỗi ứng dụng AI chỉ cần tích hợp MCP ở phía client một lần, và mỗi công cụ hoặc nguồn dữ liệu chỉ cần triển khai MCP ở phía server một lần (xem Hình 5). Nhờ đó, tất cả mọi người trong cuộc trò chuyện đều “dùng chung một ngôn ngữ”. Khi cần kết nối một AI mới với một công cụ mới, chúng ta không cần viết thêm code để integrate các tools vào mô hình mới này vì chúng đã có thể “giao tiếp với nhau” thông qua MCP.



Hình 5: MCP giải quyết bài toán tích hợp tools vào các mô hình AI

II.2. Kiến trúc của MCP

Về bản chất, MCP vận hành theo kiến trúc client-server, trong đó một ứng dụng chủ (host) có thể kết nối đến nhiều máy chủ (server) khác nhau thông qua các clients (xem Hình 6). Cấu trúc này bao gồm các thành phần sau:



Hình 6: General MCP Architecture

- **MCP Host:** Là các chương trình mà người dùng sử dụng như Claude Desktop, các IDE (Cursor, Codeium, Windsurf, v.v.) hoặc các ứng dụng AI có nhu cầu truy cập dữ liệu thông qua MCP.
- **MCP Client:** Là các clients tuân theo giao thức MCP. Mỗi MCP client chỉ giữ kết nối 1:1 với một MCP duy nhất.
- **MCP Server:** Là những chương trình nhẹ, chịu trách nhiệm cung cấp các năng lực chuyên biệt thông qua giao thức MCP.
- **Nguồn dữ liệu cục bộ (Local Data Sources):** Bao gồm tệp tin, cơ sở dữ liệu và dịch vụ đang chạy trên máy tính của chúng ta. Đây là một loại database mà các MCP servers có thể truy cập khi có sự đồng ý của người dùng.

- **Dịch vụ từ xa (Remote Services):** Là các hệ thống bên ngoài có thể kết nối qua Internet (thường qua API), và MCP server cũng có thể truy cập những dịch vụ này.

Kiến trúc lõi của MCP gồm ba thành phần là: Host, MCP client, và MCP server. Trong đó, giao tiếp giữa MCP client và MCP server được xử lý bởi transport layer và các messages truyền đi đều tuân theo chuẩn [JSON-RPC 2.0](#). Để hỗ trợ triển khai MCP server linh hoạt trong nhiều tình huống khác nhau, MCP hỗ trợ các cơ chế truyền tải như sau:

1. **Stdio Transport:** Giao tiếp thông qua luồng chuẩn đầu vào/đầu ra (standard i/o) và phù hợp cho các tiến trình chạy cục bộ trên Host. Ưu điểm của cơ chế truyền tải này là đơn giản, hiệu quả, không yêu cầu kết nối mạng.
2. **Streamable HTTP Transport:** Sử dụng giao thức HTTP để truyền dữ liệu giữa client và server. Phương thức này hỗ trợ Server-Sent Events (SSE) để truyền dữ liệu dạng streaming từ server về client. Các yêu cầu từ client đến server được gửi qua HTTP POST.

Trong [Hình 6](#), kết nối giữa các MCP Client ở Host và MCP Server 4 là dùng Streamable HTTP Transport, còn kết nối giữa các MCP Client và MCP Server 1, MCP Server 2, và MCP Server 3 có thể dùng Stdio Transport hoặc Streamable HTTP Transport (khi đó, IP address của MCP Server là `http://localhost`).

II.2.1. Host

Host là ứng dụng AI của người dùng, nơi mà các mô hình AI hoạt động và tương tác trực tiếp với chúng ta. Host có thể là các môi trường lập trình (IDE) tích hợp AI như Cursor, Windsurf, GitHub Copilot, v.v., hoặc là ứng dụng chatbot như ChatGPT và Claude Desktop, hoặc là ứng dụng trợ lý ảo tích hợp AI.

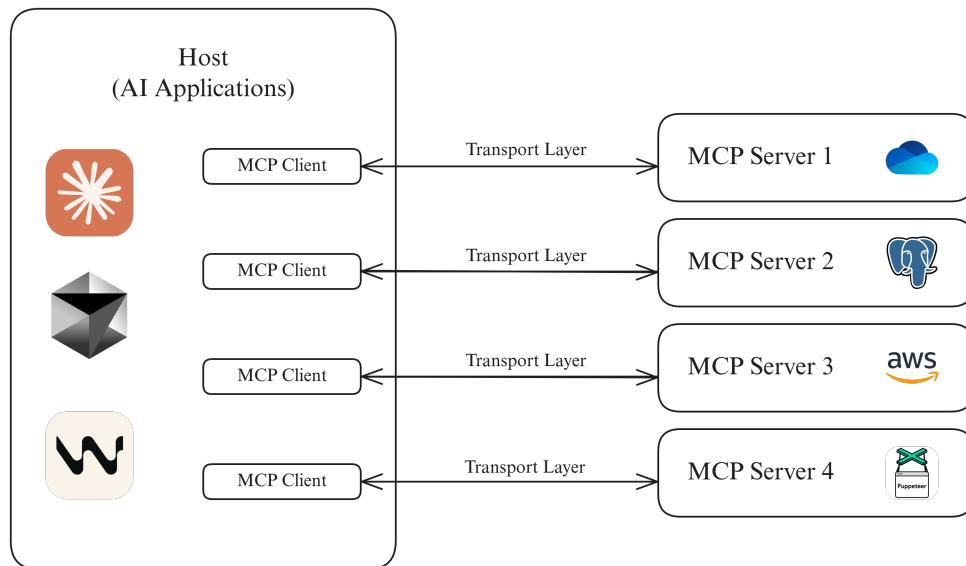
Trong hệ thống MCP, Host đóng vai trò là thành phần chủ động khởi tạo kết nối đến các MCP server khi cần. Host thực hiện các nhiệm vụ sau:

- Tạo và quản lý nhiều clients hoạt động song song, từ đó kết nối tới nhiều MCP servers cùng một lúc.
- Kiểm soát quyền kết nối và vòng đời của từng client.
- Áp dụng các chính sách bảo mật cũng như các yêu cầu liên quan đến quyền truy cập dữ liệu.
- Xử lý các quyết định ủy quyền từ phía người dùng khi một client yêu cầu quyền truy cập đến dữ liệu cá nhân của người dùng.
- Điều phối quá trình tích hợp mô hình AI/LLM, bao gồm quá trình sinh ra phản hồi khi có lệnh từ phía MCP server (sampling).

II.2.2. MCP Client

MCP client là một thành phần trung gian đóng vai trò "phiên dịch", hoạt động bên trong ứng dụng Host. Nếu ví Host như "bộ não" – nơi đưa ra quyết định và chỉ đạo hành động – thì MCP client chính là "người thông dịch", giúp truyền tải ý định của Host theo ngôn ngữ của giao thức MCP đến server, đồng thời mang kết quả trả lại.

Mỗi MCP client thiết lập và duy trì kết nối 1:1 với một MCP server duy nhất. Khi Host cần làm việc với nhiều services khác nhau (ví dụ: lưu trữ tệp, truy vấn cơ sở dữ liệu, phân tích dữ liệu), nó sẽ khởi tạo từng MCP client riêng biệt cho từng MCP server (xem Hình 7). Cách thiết kế này giúp cô lập từng kết nối, ngăn chặn rò rỉ dữ liệu và đảm bảo các sessions không bị chồng chéo.



Hình 7: Host, MCP Clients và MCP Servers

Khi một MCP client kết nối đến MCP server, quá trình bắt đầu bằng một bước handshake – tức là thương lượng giao thức giữa hai bên. Tại đây, client và server thống nhất về các tính năng được hỗ trợ, bao gồm: cách thức mã hóa thông tin, định dạng truyền và nhận tin nhắn, các công cụ mà MCP server cung cấp,... Sau khi thương lượng thành công, cả hai bên sẽ duy trì kết nối một cách liên tục để phục vụ việc giao tiếp sau đó.

Trong suốt phiên làm việc, MCP client thực hiện các nhiệm vụ sau:

- Truyền request từ host đến MCP server,
- Nhận phản hồi từ Server và gửi ngược về Host,
- Quản lý các đăng ký hoặc thông báo theo thời gian thực, ví dụ khi server chủ động gửi cập nhật cho Host.

Tóm lại, ứng dụng host giữ vai trò điều phối, quyết định khi nào và sử dụng MCP client nào cho từng tác vụ cụ thể. Trong khi đó, mỗi MCP client đảm nhiệm trọn vẹn quy trình kết nối và tương tác với MCP server, bao gồm việc thiết lập kết nối, giao tiếp theo chuẩn MCP, bảo mật và duy trì đối thoại với MCP server. Nhờ vậy, host có thể tập trung vào quá trình tương tác với người dùng, trong khi các MCP client lo toàn bộ phần tương tác với MCP server phía dưới.

II.2.3. MCP Server

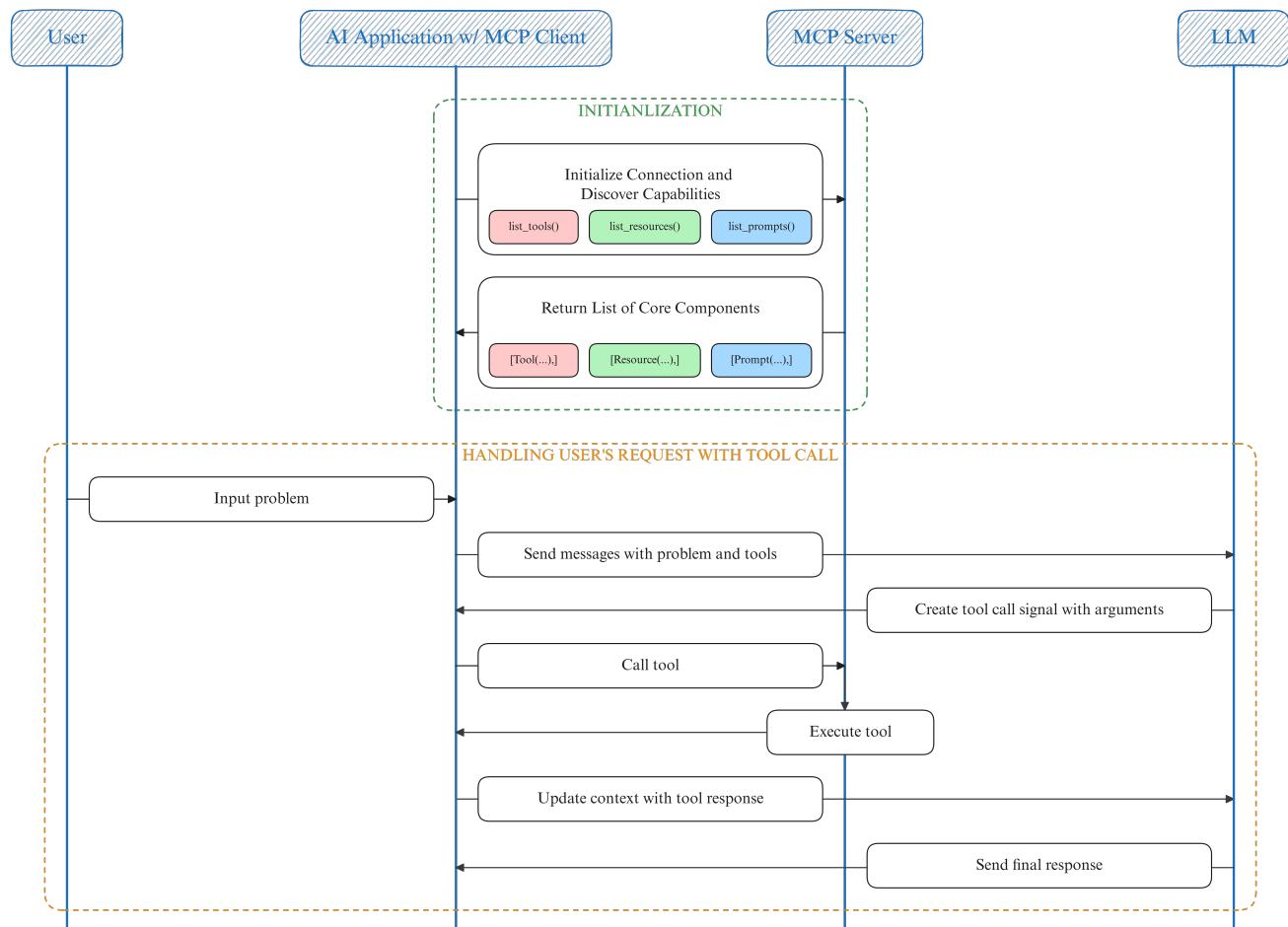
MCP server là một dịch vụ ngoài (không nằm chung với Host), đóng vai trò như một wrapper hoặc bộ chuyển đổi (adapter) cho các kho dữ liệu cục bộ, cơ sở dữ liệu, API services, cloud services, v.v.. MCP server đóng gói các khả năng chuyên biệt như công cụ, kho dữ liệu, hoặc

hành động, và công bố chúng thông qua một giao diện chuẩn hóa do MCP định nghĩa. Với cấu trúc đồng nhất, MCP server cho phép các Host dễ dàng khám phá, gọi thực thi, và nhận kết quả trả về theo định dạng chuẩn.

Thông thường, MCP server cung cấp ba loại tài nguyên mà MCP client có thể sử dụng như sau:

1. **Resources (Tài nguyên dữ liệu):** Đại diện cho ngữ cảnh dữ liệu – chẳng hạn như nội dung file, bản ghi cơ sở dữ liệu, hoặc metadata của tài liệu.
2. **Tools:** Là các hàm có thể gọi được – ví dụ như: `queryCustomers()`, `readFile()`, `findTickets()`, hoặc `execute_query()`.
3. **Prompts:** Là các mẫu tin nhắn được định nghĩa sẵn, dùng để hướng dẫn mô hình trong quá trình suy luận hoặc gọi lệnh.

II.2.4. MCP Workflow



Hình 8: MCP Workflow

Khi chúng ta khởi động AI application (vd như Claude Desktop), chúng ta sẽ khởi tạo kết nối tới MCP server và lấy những tài nguyên mà MCP server cung cấp, bao gồm tools, resources và prompts.

Khi người dùng gửi yêu cầu bài toán, AI application sẽ gửi thông tin bài toán và danh sách các tài nguyên của MCP server đến LLM để LLM quyết định có gọi tool hoặc dùng một tài nguyên khác. Nếu LLM có sử dụng tool, lúc này LLM sẽ gửi về ứng dụng AI của chúng ta gồm tên tool cần chạy và các arguments. AI application sẽ gửi yêu cầu chạy tool đến MCP server, MCP server chạy tool và trả kết quả cho AI application. Khi nhận được kết quả chạy tool, AI application sẽ update context để đưa kết quả vào conversation và gửi đến LLM để LLM quyết định bước tiếp theo. Quá trình này có thể lặp đi lặp lại cho đến khi LLM sinh ra kết quả cuối cùng.

II.3. MCP Client Features

II.3.1. Sampling

Trong MCP, **Sampling** là cơ chế cho phép một MCP server chủ động yêu cầu MCP client sử dụng mô hình ngôn ngữ (LLM) của chính phía client để sinh ra nội dung mới – thường là văn bản. Đây là một *mô hình đảo ngược vai trò* đặc biệt: thay vì chỉ có client gọi server, thì lúc này, server gọi ngược lại client để khai thác năng lực sinh ngôn ngữ.

Cụ thể, khi một MCP server cần một câu trả lời phức tạp (như tóm tắt dữ liệu, lập kế hoạch, sáng tác câu trả lời, v.v.), thay vì tự thực hiện hoặc kết nối với LLM API bên ngoài, server có thể gửi một yêu cầu `sampling/createMessage` cho client – nội dung yêu cầu chứa các tin nhắn đầu vào (`messages`) và tham số bổ sung. Client, sở hữu mô hình ngôn ngữ đã cấu hình sẵn, sẽ thực hiện việc "sample" (tức là sinh ra câu trả lời dưới dạng văn bản) và trả kết quả về server.

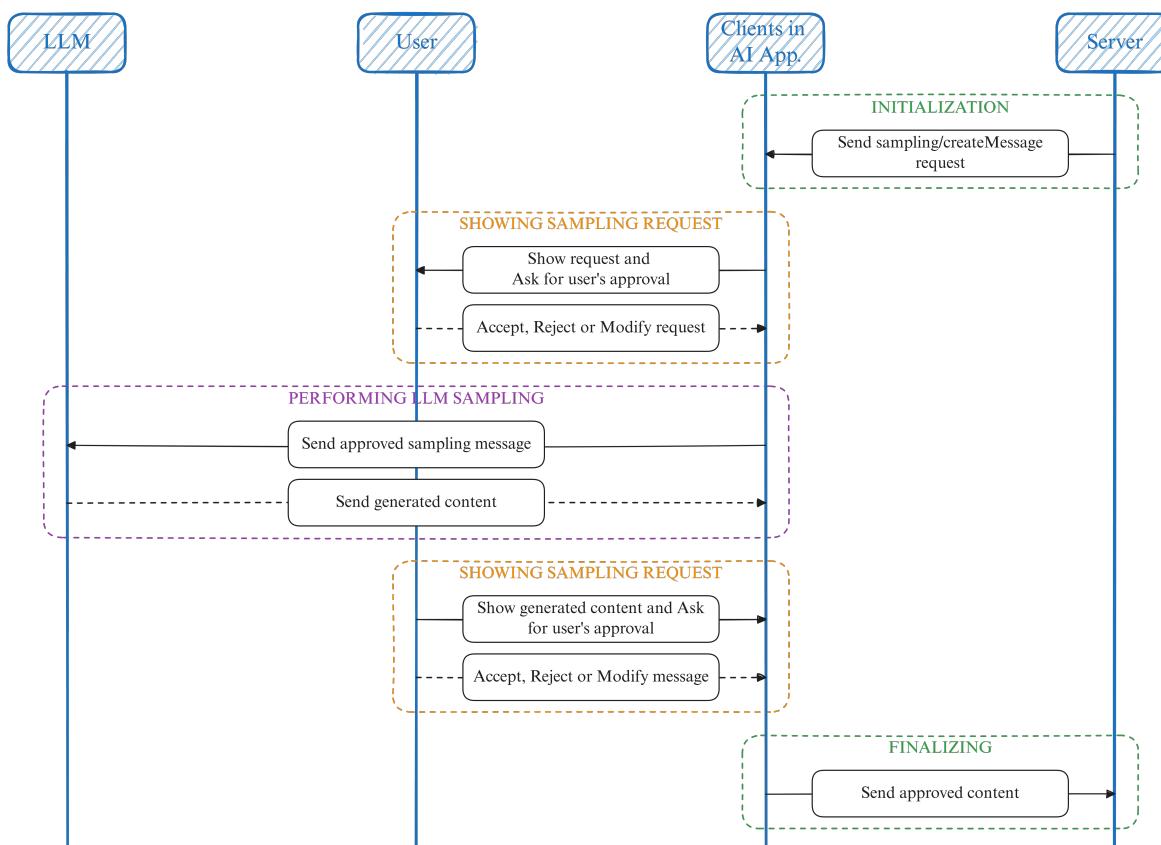
Cơ chế sampling mang lại nhiều lợi ích quan trọng như sau:

- **Tiết kiệm chi phí và hạ tầng:** MCP server không cần triển khai một mô hình AI riêng hay phải setup một luồng gọi API. Thay vào đó, MCP server sử dụng LLM của MCP client như một cách tận dụng tài nguyên đã sẵn có.
- **Tăng tính linh hoạt và agentic:** MCP server có thể tự sinh ra câu trả lời dựa trên ngữ cảnh của tình huống, ví dụ: "Tôi nên làm gì tiếp theo với dữ liệu này?", từ đó server có thể linh hoạt tương tác với người dùng giống như AI agent.
- **Giữ quyền kiểm soát cho người dùng:** Vì sampling luôn được thực hiện bởi MCP client, người dùng có thể kiểm duyệt prompt hoặc output messages trước khi nó được gửi trả về MCP server.
- **Giữ cho thiết kế của server nhẹ và dễ triển khai:** MCP server không cần cấu hình API, triển khai một LLM riêng, hay logic phức tạp mà chỉ cần biết khi nào cần nhờ "bộ não" phía client hỗ trợ quá trình xử lý thông tin.

Sampling workflow Quy trình sampling hoạt động được mô tả qua 5 bước sau (xem Hình 9):

1. Server gửi yêu cầu sampling: Thông qua RPC `sampling/createMessage`, server truyền chuỗi tin nhắn và chỉ thị để mô hình tiếp tục sinh ra câu trả lời.
2. Client đánh giá và xử lý yêu cầu: MCP client có thể hiển thị nội dung prompt cho người dùng xác nhận, từ chối hoặc chỉnh sửa nếu thấy không an toàn.
3. Gọi mô hình ngôn ngữ: Nếu được chấp thuận, client dùng LLM đã cấu hình để sinh ra nội dung tiếp theo.

4. Hậu xử lý nội dung sinh ra bởi LLM: Client có thể kiểm duyệt, cắt ngắn, lọc nội dung để đảm bảo phù hợp với đối tượng người dùng (vd như loại bỏ nội dung tiêu cực, độc hại, trích xuất keywords, tìm kiếm nội dung tương tự, v.v.).
5. Gửi kết quả về server: Câu trả lời được sinh ra từ LLM được đóng gói dưới dạng `CreateMessageResult` và gửi trả cho server để sử dụng cho các bước tiếp theo.



Hình 9: Sampling Workflow trong MCP

Ví dụ dưới đây minh họa một tool trong MCP server sử dụng sampling để viết một bài thơ với chủ đề do người dùng cung cấp:

Sampling sử dụng LLM phía client

```

1 from mcp.server.fastmcp import Context, FastMCP
2 from mcp.types import SamplingMessage, TextContent
3
4 mcp = FastMCP(name="Sampling Example")
5
6 @mcp.tool()
7 async def generate_poem(topic: str, ctx: Context) -> str:
8     """Generate a short poem about the topic using LLM sampling."""
9     prompt = f"Write a short poem about {topic}."
10    result = await ctx.session.create_message(
11        messages=[
```

```

12     SamplingMessage(
13         role="user",
14         content=TextContent(type="text", text=prompt),
15     )
16     ],
17     max_tokens=100
18 )
19     return result.content.text if result.content.type == "text" else str(result.content)
)

```

Trong đó,

- Biến ctx chứa phiên kết nối session, từ đó gửi request đến LLM ở phía client bằng method `create_message()`.
- Prompt được cấu trúc thành một `SamplingMessage` từ prompt template và tham số topi c.
- Có thể tùy chỉnh các tham số đi kèm trong quá trình sampling như `max_tokens`, `temperature`, `stop_sequences`, hoặc `includeContext`.

Sampling use cases Chúng ta có thể sử dụng sampling cho các trường hợp sau:

- **Tóm tắt kết quả:** Sau khi tool lấy dữ liệu, server có thể yêu cầu LLM tạo tóm tắt: “Tóm tắt dữ liệu sau...”.
- **Lập kế hoạch tác vụ:** Trong chuỗi agent, sampling được dùng để hỏi LLM “Tiếp theo nên làm gì?” sau mỗi bước xử lý.
- **Tạo câu trả lời tự nhiên:** Các tool như `explain_concept()` có thể dùng LLM sinh lời giải thích thay vì hard-code.
- **Đàm phán giữa agents:** Một MCP server dùng AI agent có thể dùng sampling để mô phỏng phản ứng của agent khác dựa trên prompt do mình xây dựng.
- **Phân tích kết quả tìm kiếm:** Sau khi gọi tool tìm kiếm, server có thể dùng sampling để tổng hợp hoặc đưa khuyến nghị dựa trên nội dung tìm được.

Tóm lại, sampling là cơ chế chiến lược của MCP client, cho phép MCP server kết hợp năng lực của LLM bên phía MCP client để tạo nên các quy trình AI linh hoạt, mở rộng và mang tính tác tử (agentic). Nhờ vậy, các AI agents có thể thực hiện tác vụ phức tạp mà không cần triển khai một mô hình ngôn ngữ riêng trong server.

II.3.2. Elicitation

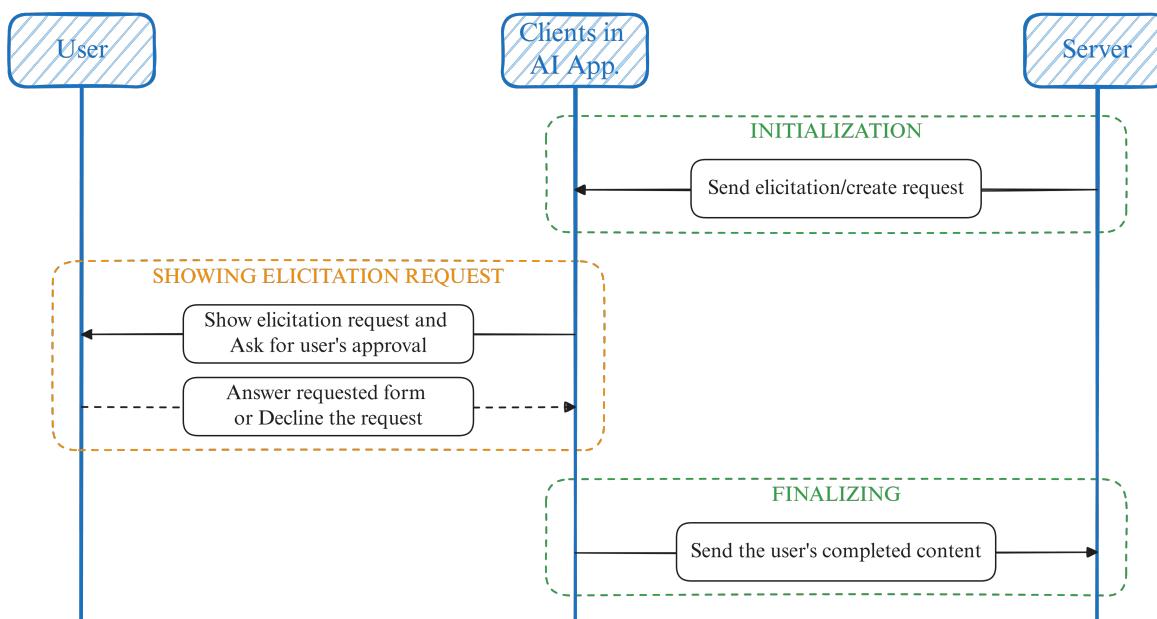
Trong quá trình tương tác giữa Host, MCP client và MCP server, có những tình huống mà server cần thêm thông tin từ người dùng để tiếp tục xử lý một tác vụ. Trong những trường hợp đó, cơ chế **elicitation** (gọi hỏi) của MCP được sử dụng. Đây là tính năng cho phép MCP server tạm dừng thực thi một công cụ, chủ động yêu cầu client hiển thị một biểu mẫu tương tác cho người dùng, và chờ người dùng điền bổ sung thông tin cần thiết rồi mới tiếp tục quá trình thực thi.

Khác với cách truyền thống là server phải nhận đủ toàn bộ input ngay khi gọi method, cơ chế

elicitation tạo ra khả năng tương tác theo nhiều lượt (multi-turn interaction), từ đó giúp các workflow trở nên linh hoạt và thân thiện hơn với người dùng. Tính năng này đặc biệt hữu ích trong các công cụ phức tạp, cần xác thực thông tin hoặc xác nhận hành vi của người dùng tại thời điểm sử dụng.

Elicitation Workflow Luồng hoạt động của một phiên elicitation tuân theo các bước sau (minh họa trong Hình 10):

1. **Server gửi yêu cầu elicitation:** MCP server sử dụng một lời gọi JSON-RPC đặc biệt (`elicitation/create`) kèm theo một thông điệp hướng dẫn người dùng và một schema định nghĩa cấu trúc dữ liệu mong đợi.
2. **Client hiển thị message:** MCP client tiếp nhận yêu cầu và hiển thị một biểu mẫu tương tác đến người dùng (trong Host), với các fields cần nhập được hiển thị tự động dựa trên schema.
3. **Người dùng phản hồi:** Người dùng có thể chọn điền thông tin và xác nhận (accept), từ chối (decline), hoặc hủy bỏ thao tác (cancel).
4. **Client gửi kết quả về server:** Nếu người dùng xác nhận, dữ liệu sẽ được validate theo schema và gửi lại cho server. Ngược lại, nếu từ chối hoặc hủy, response sẽ mang thông tin tương ứng.
5. **Server tiếp tục xử lý:** MCP server nhận kết quả và quyết định tiếp tục thực thi tool, hủy bỏ, hoặc chuyển hướng theo logic xử lý.



Hình 10: Elicitation Workflow trong MCP

Elicitation use cases Chúng ta có một số tình huống áp dụng elicitation như sau:

- Thu thập thông tin còn thiếu: ví dụ khi người dùng không cung cấp đủ tham số, cung cấp thêm thông tin cho search engine, nhập email mới khi email đã điền bị trùng, v.v.

- Làm rõ ý định: như xác định username nào khi có nhiều người trùng tên.
- Xác nhận hành động nguy hiểm: ví dụ xác nhận xóa tệp hoặc gửi email hàng loạt.
- Giao tiếp theo bước: như quá trình cài đặt một phần mềm không có sẵn trên máy, biểu mẫu nhiều trang, hoặc cấu hình hệ thống trong nhiều file.

Elicitation Implementation and Demo Chúng ta có ví dụ sau về cách thiết kế elicitation trong MCP server:

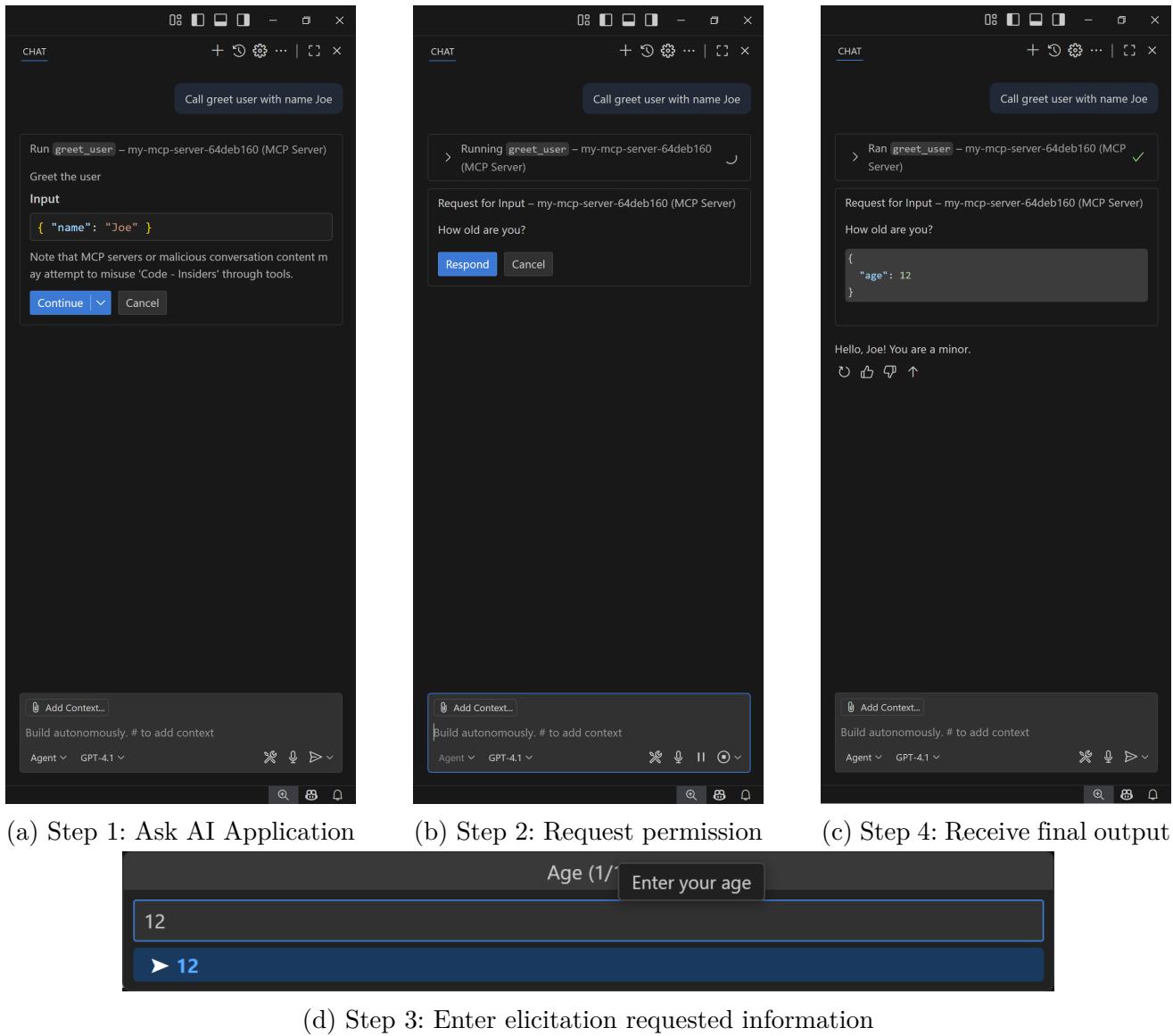
Elicitation

```

1 from pydantic import BaseModel, Field
2 from mcp.server.fastmcp import Context, FastMCP
3
4 class AgeAskingSchema(BaseModel):
5     age: int = Field(description="Enter your age", ge=0, le=120)
6
7 mcp = FastMCP(name="Server with Elicitation Example")
8
9 @mcp.tool(name="greet_user", description="Greet the user")
10 async def greet_user(name: str, ctx: Context) -> str:
11     greeting_name = f"Hello, {name}!"
12
13     asking_age_result = await ctx.elicit(
14         message="How old are you?",
15         schema=AgeAskingSchema,
16     )
17
18     if asking_age_result.action == "accept" and asking_age_result.data:
19         if asking_age_result.data.age < 18:
20             greeting_name += " You are a minor."
21         else:
22             greeting_name += " You are an adult."
23     elif asking_age_result.action == "reject":
24         greeting_name += " You chose not to provide your age."
25
26     return greeting_name
27
28 if __name__ == "__main__":
29     mcp.run(transport="streamable-http")

```

Khi tool `greet_user` được gọi, server sẽ tạo một prompt chào người dùng bằng tên đã nhập, rồi dùng cơ chế elicitation thông qua `ctx.elicit()` để hỏi thêm về độ tuổi với schema `AgeAsking Schema`. Nếu người dùng đồng ý cung cấp thông tin, server sẽ kiểm tra xem họ dưới hay trên 18 tuổi để đưa ra lời nhắn phù hợp. Nếu người dùng từ chối trả lời, tool vẫn hoạt động bình thường và thêm một câu thông báo rằng họ không muốn tiết lộ tuổi. Kết quả quá trình gọi tool `greet_user` với cơ chế elicitation được minh họa trong Hình 11. Ví dụ này minh họa rõ cách MCP sử dụng elicitation để thu thập thông tin bổ sung một cách linh hoạt, tôn trọng quyền lựa chọn của người dùng và thích ứng theo tình huống.



Hình 11: MCP Elicitation demo on Visual Studio Code Insider

Elicitation là một tính năng quan trọng trong MCP client giúp kết nối chặt chẽ hơn giữa AI application và người dùng. Cơ chế này góp phần mang lại trải nghiệm tương tác tự nhiên, linh hoạt, thay vì buộc người dùng phải đoán trước và cung cấp mọi thông tin ngay từ đầu.

II.3.3. Roots

Trong MCP, **Roots** (gốc truy cập) là một cơ chế quan trọng nhằm định nghĩa rõ phạm vi hoạt động của MCP server. Một MCP root (gọi tắt là root về sau trong bài) về bản chất là một URI (Uniform Resource Identifier) – có thể là đường dẫn tới thư mục bộ hoặc một endpoint từ xa – mà Host cung cấp để hướng dẫn MCP server biết phạm vi nào được phép truy cập và xử lý dữ liệu. Ta có ví dụ định nghĩa về roots như sau:

```
{
  "roots": [
```

```

{
    "uri": "file:///home/user/projects/myapp",
    "name": "Project Directory"
},
{
    "uri": "https://api.example.com/v1",
    "name": "API Endpoint"
}
]
}

```

Code Listing 1: Ví dụ về cấu hình roots

Ví dụ trên chỉ định rằng MCP server chỉ nên hoạt động trong thư mục dự án ở local với đường dẫn `/home/user/projects/myapp` và endpoint `APIhttps://api.example.com/v1`. Thông tin này được truyền từ MCP client đến MCP server trong quá trình khởi tạo phiến làm việc hoặc khi có thay đổi bằng cách gửi các roots đã update thông qua các routes như `roots/list` hoặc thông báo `rootsListChanged`.

Một điểm cần lưu ý là MCP không bắt buộc server phải tuân thủ tuyệt đối roots và đây là một cơ chế mang tính *thỏa thuận tin cậy* (informational contract). Tuy nhiên, các MCP server được thiết kế đúng chuẩn sẽ luôn tuân theo phạm vi roots đã được chỉ định, đặc biệt khi xử lý tài nguyên hệ thống như tệp tin hoặc endpoint nhạy cảm. Ngoài ra, roots có thể được gán thêm tên hiển thị thân thiện (friendly name), phục vụ cho UI/UX ở phía client hoặc để phân biệt trong môi trường multi-root.

Vai trò của roots bao gồm:

- **Hướng dẫn phạm vi truy cập (Scope Guidance):** MCP server sẽ biết chính xác nên tập trung vào phạm vi nào trong hệ thống tập tin hoặc các endpoint từ xa. Ví dụ, khi hỗ trợ lập trình, chỉ cần phân tích và đọc file trong thư mục dự án được chỉ định, bỏ qua toàn bộ thư mục và các file khác không liên quan (như file `.env`).
- **Giới hạn quyền truy cập (Security Boundary):** Giống như một sandbox, roots giới hạn việc MCP server truy cập ngoài vùng cho phép. Server sẽ không được phép đọc hoặc thao tác với các tài nguyên ngoài danh sách roots do client cung cấp.

Chúng ta có thể liệt kê các trường hợp sử dụng roots phổ biến như sau:

- **Coding workspace:** Hạn chế MCP server chỉ thao tác trong thư mục workspace của người dùng, ví dụ như các thư mục `src` hay `source` của dự án hiện tại,
- **Workspace chứa tài liệu:** Chỉ định thư mục chứa tài liệu Word, Markdown, Notion API workspace, hoặc các tài liệu nội bộ khác.
- **Endpoint API:** giới hạn các API được gọi của MCP server.
- **Multi-root:** Hỗ trợ người dùng làm việc trên nhiều vùng dữ liệu hoặc dự án song song (ví dụ: hai repo Git khác nhau).

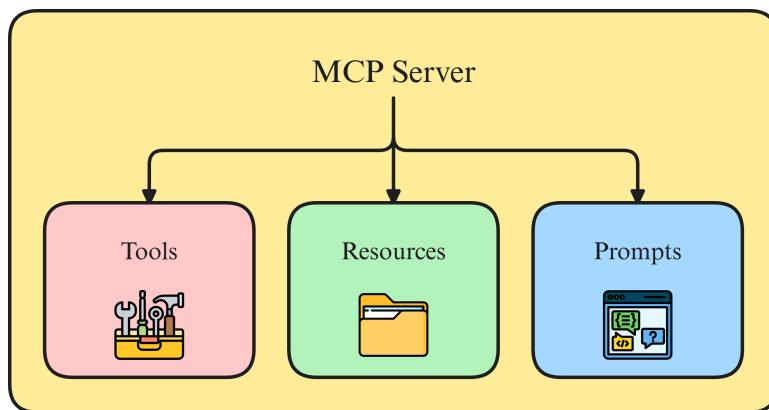
Tóm lại, roots trong MCP client là một cơ chế bảo vệ và định hướng quan trọng, vừa đảm bảo tính an toàn truy cập, vừa giúp AI agent hoạt động đúng ngữ cảnh. Việc sử dụng roots đúng cách chính là cách để triển khai nguyên tắc **least privilege** – chỉ cấp quyền vừa đủ cho tác vụ

cần thiết.

II.4. MCP Server Features

MCP server đóng vai trò là nền tảng cơ bản để bổ sung ngữ cảnh cho các mô hình AI thông qua ba thành phần cơ bản là tools, resources và prompts. Chúng ta có thể tóm gọn miêu tả về từng thành phần cơ bản như sau:

- **Tools:** Các hàm có thể thực thi, cho phép mô hình thực hiện hành động hoặc truy xuất thông tin.
- **Resources:** Dữ liệu có cấu trúc hoặc nội dung bổ sung ngữ cảnh cho mô hình,
- **Prompts:** Các template hoặc hướng dẫn được định nghĩa sẵn nhằm định hướng cách mô hình ngôn ngữ phản hồi,



Hình 12: MCP Server core primitives

Trong phần này, chúng ta sẽ vừa tìm hiểu về các thành phần cơ bản của một MCP server, vừa xây dựng một MCP server hỗ trợ quản lý sách với thư viện [MCP](#).

Đầu tiên, ta có một class `LibraryManagement` để quản lý những quyền sách trong thư viện

Class LibraryManagement

```

1 class LibraryManagement:
2     def __init__(self, books_path: Path):
3         self.books_path = books_path
4         if not books_path.exists():
5             books_path.write_text("[]", encoding="utf-8")
6             self.books = json.loads(books_path.read_text(encoding="utf-8"))
7
8     def save_books(self):
9         self.books_path.write_text(json.dumps(self.books, indent=4), encoding="utf-8")
10
11    def add_book(self, book: Book) -> str:
12        if any(b["isbn"] == book.isbn.strip() for b in self.books):
13            return f"Book with ISBN '{book.isbn}' already exists."
  
```

```
14
15     if not all([book.title.strip(), book.author.strip(), book.isbn.strip()]):
16         return "Title, author, and ISBN cannot be empty."
17
18     clean_tags = [t.strip() for t in book.tags if isinstance(t, str) and t.strip()]
19     self.books.append(
20         {
21             "title": book.title.strip(),
22             "author": book.author.strip(),
23             "isbn": book.isbn.strip(),
24             "tags": clean_tags,
25         }
26     )
27     self.save_books()
28     return f"Book '{book.title}' by {book.author} added to the library."
29
30 def remove_book(self, isbn: str) -> str:
31     updated = [b for b in self.books if b["isbn"] != isbn.strip()]
32     if len(updated) == len(self.books):
33         return f"No book found with ISBN '{isbn}'."
34     self.books = updated
35     self.save_books()
36     return f"Book with ISBN '{isbn}' removed from the library."
37
38 def get_num_books(self) -> int:
39     return len(self.books)
40
41 def get_all_books(self) -> list:
42     return self.books
43
44 def get_book_by_index(self, index: int) -> dict:
45     if 0 <= index < len(self.books):
46         return self.books[index]
47     return {"error": "Book not found."}
48
49 def get_book_by_isbn(self, isbn: str) -> dict:
50     for b in self.books:
51         if b["isbn"] == isbn.strip():
52             return b
53     return {"error": "Book not found."}
54
55 def get_suggesting_random_book_prompt(self) -> str:
56     return "Suggest a random book from the library. The suggestion should include  
the title, author, and a brief description.  
"
57
58 def get_suggesting_book_title_by_abstract_prompt(self, abstract: str) -> str:
59     return f"Suggest a memorable, descriptive title for a book based on the  
following abstract: {abstract}"
60
61 def get_analyzing_book_messages(self, book: dict, query: str) -> list[dict[str, str]]:
62     return [
63         {
```

```

64         "role": "user",
65         "content": "This is the book I want to analyze: " + json.dumps(book),
66     },
67     [
68         {
69             "role": "assistant",
70             "content": "Sure! Let's analyze this book together. What would you like
71             to know?",
72         },
73         {"role": "user", "content": query},
74     ]

```

Class này có các methods sau:

- **save_books**: Ghi toàn bộ danh sách sách hiện tại vào tệp JSON, đảm bảo mọi thay đổi (thêm hoặc xoá sách) được lưu lại một cách nhất quán.
- **add_book**: Thêm một quyển sách mới vào thư viện. Hàm sẽ kiểm tra tính hợp lệ của các trường dữ liệu (title, author, ISBN) và tránh trùng lặp ISBN. Nếu thành công, dữ liệu sách được ghi vào danh sách và cập nhật vào tệp lưu trữ.
- **remove_book**: Xoá một quyển sách dựa trên ISBN. Nếu không tìm thấy sách phù hợp, hàm trả về thông báo lỗi. Nếu xoá thành công, cập nhật lại dữ liệu lưu trữ.
- **get_num_books**: Trả về tổng số sách hiện có trong thư viện.
- **get_all_books**: Trả về toàn bộ danh sách sách dưới dạng một danh sách Python.
- **get_book_by_index**: Lấy thông tin một quyển sách theo chỉ số (index) trong danh sách. Nếu chỉ số không hợp lệ, trả về thông báo lỗi.
- **get_book_by_isbn**: Trả về thông tin của một quyển sách dựa trên ISBN. Nếu không tìm thấy, trả về thông báo lỗi.
- **get_suggesting_random_book_prompt**: Trả về một chuỗi lệnh nhắc (prompt) yêu cầu gợi ý một quyển sách ngẫu nhiên từ thư viện.
- **get_suggesting_book_title_by_abstract_prompt**: Nhận vào một đoạn abstract của quyển sách và tạo ra một prompt để gợi ý tiêu đề sách phù hợp, mang tính mô tả và ẩn tượng.
- **get_analyzing_book_messages**: Tạo ra một danh sách các tin nhắn (theo cấu trúc trò chuyện giữa user và assistant) nhằm phục vụ phân tích một quyển sách thông qua mô hình hội thoại. Cấu trúc tin nhắn phù hợp với định dạng sử dụng trong các ứng dụng như ChatGPT hay các hệ thống trò chuyện thông minh.

Dựa vào ứng dụng trên, ta sẽ thiết kế một MCP server phục vụ việc quản lý sách trong thư viện. Đi qua mỗi core component của MCP server, chúng ta sẽ điền dần vào đoạn code dưới đây:

MCP Server implementation mà chúng ta sẽ điền vào

```

1 import json
2 from pathlib import Path
3 from typing import Dict, Sequence

```

```
4
5 from mcp.server.lowlevel import Server
6 from mcp.server.stdio import stdio_server
7 from mcp.types import (
8     EmbeddedResource,
9     GetPromptResult,
10    ImageContent,
11    Prompt,
12    PromptArgument,
13    PromptMessage,
14    Resource,
15    ResourceTemplate,
16    TextContent,
17    Tool,
18 )
19 from pydantic import BaseModel, Field
20 from pydantic.networks import AnyUrl
21
22 class LibraryManagement:
23     ...
24
25     async def serve() -> None:
26         books_path = Path("books.json")
27         library = LibraryManagement(books_path)
28
29         server = Server("mcp-library")
30
31         ##### TOOLS #####
32         @server.list_tools()
33         async def list_tools() -> list[Tool]:
34             """List all available tools for the library management system."""
35             ...
36
37         @server.call_tool()
38         async def call_tool(
39             name: str, arguments: dict
40         ) -> Sequence[TextContent | ImageContent | EmbeddedResource]:
41             """Call a specific tool by name with the provided arguments."""
42             ...
43
44         ##### RESOURCES #####
45         @server.list_resources()
46         async def list_resources() -> list[Resource]:
47             """List all available resources."""
48             ...
49
50         @server.list_resource_templates()
51         async def list_resource_templates() -> list[ResourceTemplate]:
52             """List all available resource templates."""
53             ...
54
55         @server.read_resource()
56         async def read_resource(uri: AnyUrl) -> str:
57             """Read a resource by its URI."""
```

```

58     ...
59
60     ##### PROMPTS #####
61     @server.list_prompts()
62     async def list_prompts() -> list[Prompt]:
63         """List all available prompts."""
64         ...
65
66     @server.get_prompt()
67     async def get_prompt(name: str, arguments: Dict[str, str] | None) ->
68         GetPromptResult:
69         """Get a specific prompt by name."""
70         ...
71
72     options = server.create_initialization_options()
73     print(f"Server is running with options: {options}")
74
75     async with stdio_server() as (read_stream, write_stream):
76         await server.run(read_stream, write_stream, options)
77
78 if __name__ == "__main__":
79     import asyncio
80
81     asyncio.run(serve())

```

II.4.1. Tools

Tools trong MCP Server là một thành phần cốt lõi, cho phép các mô hình AI thực hiện các hành động thực tế thông qua việc tương tác với các hàm được thiết kế để thực thi ở phía server.

Giống như resources, tools được định danh bằng các unique name và có thể kèm theo mô tả để hướng dẫn mô hình AI chọn đúng để thực thi. Tuy nhiên, khác với resources, tools đại diện cho các thao tác động thông qua việc chúng có khả năng thay đổi trạng thái, thông tin của MCP server hoặc tương tác với các hệ thống bên ngoài khác.

Chúng ta có các ví dụ thực tế về tool ở các MCP server sau:

- **Google Drive MCP Server:** cung cấp tool `search(query)` để tìm kiếm các file tương ứng với `query` trong Google Drive,
- **Git MCP Server:** có các tool quản lý và tương tác với các Git repo như `git_init(repo_path)` nhằm khởi tạo repo, `git_status(repo_path)` để hiển thị trạng thái repo hiện tại (giống lệnh `git status`), `git_commit(repo_path, message)` phục vụ việc lưu lại các thay đổi vào kho Git, `git_add(repo_path, files)` hỗ trợ việc đưa nội dung các tệp vào khu vực staging, và `git_checkout(repo_path, branch_name)` để chuyển sang một branch khác. Tuy nhiên, MCP Server này không cung cấp Resource hay Prompts,
- **Time MCP Server:** lấy thông tin thời gian hiện tại và chuyển đổi múi giờ thông qua hai tool sau: `get_current_time(timezone)` để lấy thời gian hiện tại tại một múi giờ cụ thể hoặc tại múi giờ hệ thống, và `convert_time(source_timezone, time, target_timezone)` để chuyển đổi thời gian từ một múi giờ này sang múi giờ khác,

- **FileSystem MCP Server:** hỗ trợ thao tác hệ thống tệp như đọc/ghi file và liệt kê thư mục với các tool: `read_file(path)` để đọc nội dung tệp, `write_file(path, content)` để ghi nội dung vào tệp, và `list_directory(path)` để liệt kê các mục trong thư mục. Tất cả thao tác đều bị giới hạn trong các thư mục được cấu hình trước.
- **PostgreSQL MCP Server:** cung cấp quyền truy cập chỉ đọc vào cơ sở dữ liệu PostgreSQL và hỗ trợ tool `query(sql)` để thực thi các truy vấn SQL.

Tool trong MCP server hoạt động tương tự function calling, cho phép mô hình AI kích hoạt tính toán, điều chỉnh trạng thái, tích hợp với API bên ngoài hoặc thực hiện các hoạt động hệ thống nằm ngoài khả năng cơ bản của LLM. Điểm đặc trưng của tools trong MCP là được điều khiển bởi mô hình, nhưng chịu sự giám sát từ server và người dùng. Mô hình AI có thể tự động đề xuất việc sử dụng tool dựa trên sự hiểu biết của nó về nhiệm vụ đang xử lý (ví dụ như tìm file trong Google Drive sẽ dùng tool `search(query)` thông qua Google Drive MCP Server, đọc file từ local với tool `readFile(filename)`). Tuy nhiên, mọi yêu cầu sử dụng tool đều phải được thực thi bởi server và khi cần thiết (ví dụ như quyền truy cập vào Google Drive, quyền truy cập file, quyền thực thi command trong terminal) thì phải có sự đồng ý từ người dùng. Quy trình này đảm bảo sự kết hợp giữa sức mạnh và tính kiểm soát: mô hình AI được trao quyền mở rộng năng lực tính toán và tương tác với môi trường, nhưng những hành động quan trọng hoặc nhạy cảm sẽ luôn được giám sát chặt chẽ bởi người dùng.

Các đặc trưng của Tool trong MCP server có thể được tóm gọn như sau:

- **Khám phá (Discovery):** MCP client có thể truy xuất danh sách các công cụ hiện có bằng cách gửi request đến endpoint `tools/list`.
- **Kích hoạt (Invocation):** Các công cụ được gọi thông qua yêu cầu `tools/call`, trong đó server sẽ thực hiện thao tác được yêu cầu và trả về kết quả.
- **Tính linh hoạt (Flexibility):** Các công cụ có thể rất đa dạng, từ các phép tính đơn giản cho đến những tương tác phức tạp với API.

Luồng hoạt động của Tool Quá trình gọi một Tool và lấy kết quả của mô hình AI với MCP như sau:

1. LLM lựa chọn một tool cùng với arguments để chạy tool đó (ví dụ: tool `get_weather` với tham số là `{"location": "SanFrancisco"}`).
2. MCP client đóng vai trò trung gian, có thể yêu cầu người dùng xác nhận hành động thực thi tool và thực hiện lệnh gọi tool đến MCP server.
3. MCP server thực thi tool đó và trả về output dưới dạng có cấu trúc (structured response).
4. MCP client chuyển kết quả trở lại cho mô hình AI, giúp quá trình suy luận và giải quyết bài toán tiếp tục một cách liền mạch.

Định nghĩa Tool Mỗi tool được định nghĩa theo cấu trúc sau:

Cấu trúc định nghĩa tool trong MCP server

```
{
  name: string,
  description?: string,
  inputSchema: {
    type: "object",
    properties: { ... }
  },
  annotations?: {
    title?: string,
    readOnlyHint?: boolean,
    destructiveHint?: boolean,
    idempotentHint?: boolean,
    openWorldHint?: boolean
  }
}
```

- **name**: Tên của tool và không được trùng lặp trong cùng MCP server
- **description** (Optional): Mô tả về tool
- **inputSchema**: JSON Schema định nghĩa các tham số đầu vào của tool
 - **type**: Luôn được set giá trị "object"
 - **properties**: Các tham số cụ thể của tool
- **annotations** (tùy chọn): Gợi ý về hành vi của tool
 - **title**: Tiêu đề dễ hiểu dành cho con người
 - **readOnlyHint**: Nếu giá trị là **true**, tool không thay đổi môi trường/dữ liệu
 - **destructiveHint**: Nếu giá trị là **true**, tool có thể thực hiện các thay đổi mang tính phá hủy
 - **idempotentHint**: Nếu giá trị là **true**, gọi lặp lại với cùng tham số sẽ không gây thêm hiệu ứng
 - **openWorldHint**: Nếu giá trị là **true**, tool tương tác với các object khác ở bên ngoài

Giả sử chúng ta có hàm `read_txt_file` để đọc một file .txt từ phía local storage:

Hàm `read_txt_file` để đọc file .txt

```

1 def read_txt_file(file_path: str):
2     """
3     Read the contents of a text file.
4
5     Args:
6         file_path (str): Path to the text file.
7

```

```

8     Returns:
9         dict: Contents of the text file.
10        """
11    try:
12        with open(file_path, 'r') as file:
13            content = file.read()
14        return {"content": content}
15    except Exception as e:
16        return {"error": str(e)}

```

chúng ta định nghĩa tool `read_txt_file` trong MCP server tương ứng với hàm trên như sau:

```

{
    "name": "read_txt_file",
    "title": "File Reader",
    "description": "Read the contents of a text file",
    "inputSchema": {
        "type": "object",
        "properties": {"file_path": {"type": "string", "description": "Path to the text file"}},
        "required": ["file_path"],
    },
    "outputSchema": {
        "type": "object",
        "properties": {
            "content": {"type": "string", "description": "Contents of the text file"},
        },
        "required": ["content"],
    },
}

```

Xây dựng Tools cho Library MCP Server Để xây dựng tools cho Library MCP Server, chúng ta viết hai methods sau:

- `list_tools()` dùng để liệt kê các tools mà MCP server hỗ trợ
- `call_tool(name, arguments)` để MCP server thực thi tool khi MCP client gọi tool đó.

Chúng ta sẽ thêm 3 tools sau đây vào Library MCP Server:

- `add_book` để thêm sách vào thư viện
- `remove_book`: xóa một quyển sách khỏi thư viện
- `get_num_books`: lấy thông tin về số sách hiện có trong thư viện

Với hàm `list_tools`, ta sử dụng decorator `@server.list_tools()` và định nghĩa từng tool trong hàm như sau:

Thiết kế hàm list_tools

```

1 class Book(BaseModel):
2     title: str = Field(..., description="The title of the book")
3     author: str = Field(..., description="The author of the book")
4     isbn: str = Field(..., description="The ISBN of the book")
5     tags: list[str] = Field(default_factory=list, description="Tags associated with the
6                               book")
7
8 class BookISBNInput(BaseModel):
9     isbn: str = Field(..., description="The ISBN of the book to be removed or retrieved
10                           ")
11
12
13 ...
14
15 @server.list_tools()
16 async def list_tools() -> list[Tool]:
17     return [
18         Tool(
19             name="add_book",
20             description="Add a book to the library",
21             inputSchema={
22                 "type": "object",
23                 "required": ["title", "author", "isbn"],
24                 "properties": {
25                     "title": {
26                         "description": "The title of the book",
27                         "type": "string",
28                     },
29                     "author": {
30                         "description": "The author of the book",
31                         "type": "string",
32                     },
33                     "isbn": {
34                         "description": "The ISBN of the book",
35                         "type": "string",
36                     },
37                     "tags": {
38                         "description": "Tags associated with the book",
39                         "items": {"type": "string"},
40                         "type": "array",
41                     },
42                 },
43             ),
44         Tool(
45             name="remove_book",
46             description="Remove a book by its ISBN",
47             inputSchema=BookISBNInput.model_json_schema(),
48         ),
49         Tool(
50

```

```

51         name="get_num_books",
52         description="Get the total number of books",
53         inputSchema={"type": "object", "properties": {}},
54     ),
55 ]

```

Trong đó, các tool được định nghĩa bằng class Tool với arguments `name`, `description`, và `inputSchema`. Đối với tool `add_book`, input schema là thông tin một quyển sách (gồm tên, author, ISBN, và tags) được viết dưới dạng JSON object. Các field của JSON object này được định nghĩa giống như cách định nghĩa field ở MCP server đã đề cập ở phần II.4.1.. Đối với tool `remove_book`, ta có input schema là data class `BookISBNInput`. Với tool `get_num_books`, ta không có input arguments nên input schema rỗng.

Chúng ta tiếp tục implement hàm `call_tool` với decorator `@server.call_tool()` như sau:

Thiết kế hàm call_tool

```

1 @server.call_tool()
2 async def call_tool(
3     name: str, arguments: dict
4 ) -> Sequence[TextContent | ImageContent | EmbeddedResource]:
5     try:
6         match name:
7             case "add_book":
8                 book = AddBookInput(**arguments)
9                 result = library.add_book(book)
10                return [TextContent(type="text", text=result)]
11
12             case "remove_book":
13                 data = BookISBNInput(**arguments)
14                 result = library.remove_book(data.isbn)
15                return [TextContent(type="text", text=result)]
16
17             case "get_num_books":
18                 result = library.get_num_books()
19                return [TextContent(type="text", text=str(result))]
20
21             case _:
22                 raise ValueError(f"Unknown tool: {name}")
23
24     except Exception as e:
25         raise ValueError(f"LibraryServer Error: {str(e)}")

```

Chúng ta có thể thấy rằng hàm `call_tool` đóng vai trò như một cầu nối giữa các lệnh được gọi từ phía MCP client với các hàm của hệ thống quản lý thư viện. Cụ thể như sau:

- Sau khi match được tên tool, ta giải mã input arguments về dạng schema đã định nghĩa (`AddBookInput`, `BookISBNInput`, v.v.), sau đó truyền các giá trị này vào hàm tương ứng của class `LibraryManagement`.
- Kết quả trả về được chuẩn hoá dưới dạng một danh sách các object `TextContent`. Đây là

cấu trúc được định nghĩa để trả lại phản hồi dưới dạng văn bản.

II.4.2. Resources

Resources trong MCP server là các tài nguyên dữ liệu chỉ đọc mà server cung cấp cho client, ví dụ như các file code, nhật ký (logs), bản ghi cơ sở dữ liệu, hình ảnh, video, audio, hoặc phản hồi từ API. Đối với các mô hình AI, resources đóng vai trò như ngữ cảnh bổ sung kiến thức. Chúng là những thông tin hữu ích có thể được truy xuất từ MCP server và thêm vào prompt trong quá trình giao tiếp với mô hình AI, nhưng resource không bao giờ bị chỉnh sửa/thay đổi.

Không giống như tool hay function call sẽ thực thi các đoạn mã và có thể tạo ra tác động đến server, resources được thiết kế hoàn toàn để phục vụ việc truy xuất thông tin, giúp chúng trở nên an toàn và dễ dự đoán kết quả nào được trả về trong quá trình tìm kiếm. Resource đặc biệt phù hợp trong những tình huống mà mô hình cần hiểu ngữ cảnh hơn là thực thi một hành động cụ thể. Ví dụ, khi người dùng nói: "Hiển thị số liệu doanh thu mới nhất," client phía Host có thể truy xuất một tệp CSV hoặc bản chụp dữ liệu từ cơ sở dữ liệu thông qua một Resource và chèn nó vào đầu prompt, nhưng sẽ không thực thi bất kỳ hành động nào để tránh việc tác động vào dữ liệu ở server.

Key Features

- Kiểm soát bởi ứng dụng:** MCP client có toàn quyền quyết định khi nào và bằng cách nào để truy xuất resources. Một MCP client có thể cho phép người dùng lựa chọn resource từ danh sách, trong khi một MCP client khác có thể tự động chọn resources dựa trên các keywords trong messages của người dùng.
- Đặt tên linh hoạt và dễ dàng truy xuất:** Các resources được định danh thông qua URI với các giao thức tùy chỉnh (custom protocol) như `file://`, `mysql://`, `screen://`, v.v.. Client có thể truy xuất danh sách resource thông qua endpoint `resources/list` và sử dụng các mẫu URI (URI templates) để thêm các arguments (ví dụ: `log://{{date}}.log` có argument là `date`, phục vụ cho việc truy xuất log theo ngày).
- Đa dạng loại nội dung:** Resources của MCP server có hai dạng, là dạng văn bản (mã hóa UTF-8) như source code, file config, file logs, dữ liệu JSON/XML, text thuần túy, hoặc dữ liệu dưới dạng mã nhị phân (được mã hóa dưới dạng base64) như hình ảnh, âm thanh, video, PDFs, v.v..
- Cập nhật theo thời gian thực:** Client có thể đăng ký nhận thông báo khi danh sách resources bị thay đổi thông qua endpoint `notifications/resources/list_changed`, hoặc là nội dung của một resource cụ thể được cập nhật (through qua endpoint `resources/subscribe` và endpoint `notifications/resources/updated`)

Resource URIs Resources được định danh bằng các URI theo định dạng sau:

[protocol]://[host]/[path]

trong đó, `protocol` và `path` được xác định bởi cách triển khai của MCP . Các server có thể tự định nghĩa các scheme URI tùy chỉnh của riêng mình. Ta có các ví dụ URI của Resource như sau:

- `csv://datasets/sales/2025.csv`

- mysql://database/customers/schema
- log://datetime.log

và các resource được lấy ở các MCP server thực tế như sau:

- **Google Drive MCP Server**: resource `gdrive:///<file_id>` hỗ trợ client truy cập file ở Google Drive với `file_id`.
- **SQLite MCP Server**: cung cấp resource `memo://insights` như một bản ghi chú tổng hợp thông tin chi tiết kinh doanh được cập nhật liên tục trong quá trình phân tích của server.
- **n8n MCP Server**: là một MCP server cho phép mô hình AI tương tác trực tiếp với hệ thống workflow của n8n. n8n MCP Server cung cấp các resource sau: `n8n://workflows/list` để liệt kê toàn bộ workflows, `n8n://workflow/{id}` để truy vấn thông tin chi tiết của một workflow cụ thể, `n8n://executions/{workflowId}` để lấy danh sách các lần thực thi của một workflow, và `n8n://execution/{id}` để xem chi tiết một lần thực thi.
- **Everything MCP server**: cung cấp 100 tài nguyên mẫu để kiểm thử khả năng xử lý resource trong MCP client. Các resource có URI theo dạng `test://static/resource/{id}`.

Định nghĩa Resource trong MCP Server Resource trong MCP server được định nghĩa như sau:

Định nghĩa resource trong MCP server

```
{
  uri: string,
  name: string,
  description?: string,
  mimeType?: string,
  size?: number
}
```

trong đó,

- `uri`: Unique name (định danh) duy nhất của resource
- `name`: Tên dễ hiểu
- `description` (tùy chọn): Mô tả bổ sung
- `mimeType` (tùy chọn): Loại MIME (kiểu nội dung) của resource
- `size` (tùy chọn): Kích thước resource tính bằng byte

Ví dụ như với file PDF có url <https://docs.google.com/document/d/1ve33Kex-cJKAKBToy---lddE6BLw-LA6bBPtbTw4F8o/edit> trên Google Drive, ta có định nghĩa của file ở Google Drive MCP server như sau:

```
{
  "uri": "gdrive:///1ve33Kex-cJKAKBToy---lddE6BLw-LA6bBPtbTw4F8o",
  "name": "Example Resource",
```

```

    "description": "This is an example resource for demonstration
    ↵ purposes.",
    "mimeType": "application/pdf",
    "size": 4096,
}

```

Xây dựng Resource cho Library MCP Server Đối với Library MCP Server của chúng ta, chúng ta sẽ xây dựng cả static resource (resource tĩnh) và dynamic resource (resource động).

Đối với static resource, chúng ta viết hàm `list_resources()` và dùng decorator `@server.list_resources()` như sau:

Xây dựng hàm `list_resources()`

```

1 async def list_resources() -> list[Resource]:
2     return [
3         Resource(
4             name="all_books",
5             title="All Books",
6             uri=AnyUrl("books://all"),
7             description="Get all books in the library",
8         ),
9     ]

```

Đây là nơi chúng ta định nghĩa các resource tĩnh với các fields như `name`, `title`, `uri`, và `description`. Resource `all_books("books://all")` được định nghĩa để MCP client có thể lấy dữ liệu tất cả các quyển sách trong MCP .

Đối với dynamic resource, chúng ta implement hàm `list_resource_templates()` với decorator `@server.list_resource_templates()`:

Xây dựng hàm `list_resource_templates()`

```

1 @server.list_resource_templates()
2 async def list_resource_templates() -> list[ResourceTemplate]:
3     return [
4         ResourceTemplate(
5             name="book_by_index",
6             title="Book by Index",
7             uriTemplate="books://index/{index}",
8             description="Get a book by its index in the library",
9         ),
10        ResourceTemplate(
11            name="book_by_isbn",
12            title="Book by ISBN",
13            uriTemplate="books://isbn/{isbn}",
14            description="Get a book by its ISBN",
15        ),
16    ]

```

trong đó, `index` là argument của URI `books://index/index` để lấy thông tin quyển sách theo index trong database, và URI `books://isbn/isbn` với argument `isbn` để tìm sách theo mã ISBN. URI của dynamic resource được định nghĩa bằng field `uriTemplate` thay vì field `uri` như static resource.

Cuối cùng, chúng ta viết hàm `read_resource(uri)` với decorator `@server.read_resource()` để trả về resource dựa trên URI tương ứng:

Xây dựng hàm `read_resource(uri)`

```

1 @server.read_resource()
2 async def read_resource(uri: AnyUrl) -> str:
3     uri_str = str(uri)
4     if uri_str == "books://all":
5         books = library.get_all_books()
6         return json.dumps(books, indent=4)
7     elif uri_str.startswith("books://index/"):
8         index_str = uri_str.split("/")[-1]
9         try:
10             index = int(index_str)
11             book = library.get_book_by_index(index)
12             if "error" in book:
13                 raise ValueError(book["error"])
14             return json.dumps(book, indent=4)
15         except ValueError:
16             raise ValueError(f"Invalid index: {index_str}")
17     elif uri_str.startswith("books://isbn/"):
18         isbn = uri_str.split("/")[-1]
19         book = library.get_book_by_isbn(isbn)
20         if "error" in book:
21             raise ValueError(book["error"])
22         return json.dumps(book, indent=4)
23     else:
24         raise ValueError(f"Resource '{uri}' not found.")

```

Đây là phần xử lý thực thi khi có yêu cầu truy xuất một resource cụ thể, dựa trên URI truyền vào. Hàm `read_resource` sẽ phân tích URI và gọi hàm tương ứng từ class `LibraryManagement` để lấy resource được định nghĩa ở hai hàm liệt kê trên, sau đó tuân tự hóa thành định dạng JSON để trả về cho MCP client.

II.4.3. Prompts

Prompts trong MCP Server là các prompt template được định nghĩa sẵn hoặc mẫu cuộc trò chuyện mà MCP server cung cấp cho MCP client. Đây thực chất là các đoạn hội thoại hoặc các query template được lưu trữ trong hệ thống mà chúng ta hoặc các hệ thống tự động hoặc hệ thống tự động có thể gọi ra dùng để đơn giản hóa quá trình tương tác với mô hình AI.

Ví dụ, một server có thể cung cấp một prompt tên là `find_bug` (xem code bên dưới), chứa các lệnh hướng dẫn mô hình AI đọc và tìm bug dựa trên error message và các chỗ trống để chèn nội dung động.

```
1 @mcp.prompt()
```

```

2 def code_debug(code: str, error_message: str) -> List[Dict[str, str]]:
3     return [
4         {"role": "system", "content": f"You are a helpful assistant that helps debug code."},
5         {
6             "role": "user",
7             "content": f"Here is the code that has an error: {code}\nError message: {error_message}",
8         },
9     ]

```

Các prompt này sẽ hiển thị trong giao diện người dùng hoặc dropdown menu, cho phép người dùng lựa chọn, tùy chỉnh và chèn vào luồng hội thoại một cách linh hoạt.

Lợi ích của prompt khi đặt ở MCP Server Không giống như tools, vốn được LLM tự động gọi để thực hiện hành động, prompts là những chỉ dẫn do người dùng hoặc lập trình viên chủ động chọn lựa nhằm định hình hành vi của mô hình. Chúng bao gồm các template được thiết kế để hướng dẫn mô hình AI hoạt động - ví dụ: "Bạn là một developer chuyên debug các đoạn code lỗi ... và có thể được áp dụng trên nhiều MCP với quy mô lớn. Việc MCP server công khai các prompt mang lại các lợi ích như sau:

- Tính nhất quán: Cung cấp định dạng chuẩn cho các tác vụ lặp lại như tóm tắt, kiểm tra mã, hay gỡ lỗi.
- Khả năng tái sử dụng: Client có thể gọi server (prompts/list, prompts/get) để lấy các mẫu có cấu trúc và tích hợp vào quy trình làm việc của người dùng.
- Tính mở rộng: Prompt có thể bao gồm arguments, trả về resources, hoặc nhiều messages - hỗ trợ nội dung động và multi-turn conversations,
- Việc tách biệt prompt template ra khỏi client cho phép duy trì, chia sẻ và cập nhật các prompt tốt nhất tại MCP server mà không cần thay đổi code ở phía client.

Chúng ta có nột vài prompts trong các MCP server thực tế như sau:

- [Everything MCP Server](#): để test MCP client, MCP Server này cung cấp các prompts như `simple_prompt` và `prompt` cơ bản, `complex_prompt(temperature, style?)` để mô phỏng cuộc đối thoại nhiều lượt với hai tham số đầu vào, và `resource_prompt(resourceId)` là prompt minh họa cho việc nhúng resource vào nội dung prompt.
- [Fetch MCP server](#): hỗ trợ prompt `fetch(url)` để truy xuất nội dung từ trang web và chuyển đổi HTML thành markdown. Thích hợp cho các tác vụ đọc hiểu web, với khả năng phân trang nội dung thông qua `start_index`.
- [Sentry MCP server](#): cung cấp prompt `sentry-issue(issue_id_or_url)` để truy xuất chi tiết lỗi từ Sentry.io dưới dạng hội thoại. Kết quả bao gồm tiêu đề lỗi, mức độ nghiêm trọng, thời gian xuất hiện, số lượng sự kiện và toàn bộ stacktrace.
- [SQLite MCP server](#): cung cấp prompt `mcp-demo(topic)` để hướng dẫn người dùng thực hiện phân tích dữ liệu kinh doanh. Prompt này tạo schema và dữ liệu mẫu phù hợp, đồng thời tương tác với resource `memo://insights` (bản ghi động cập nhật liên tục các kết luận phân tích).
- [Deep Research MCP server](#): cung cấp prompt `deep-research` dành cho các nhiệm vụ

nghiên cứu chuyên sâu., hỗ trợ toàn bộ quy trình nghiên cứu từ mở rộng câu hỏi, tạo các tiểu đề tài, tìm kiếm nguồn đáng tin cậy, phân tích nội dung cho đến tổng hợp thành báo cáo đầy đủ, có trích dẫn và định dạng chuyên nghiệp.

Định nghĩa Prompt trong MCP Server Prompt trong MCP server được định nghĩa như sau:

Định nghĩa prompt trong MCP server

```
{
  name: string,
  description?: string,
  arguments?: [
    {
      name: string,
      description?: string,
      required?: boolean
    }
  ]
}
```

- **name**: Tên duy nhất của prompt (không được trùng)
- **description** (tùy chọn): Mô tả dễ hiểu về prompt dành cho con người
- **arguments** (tùy chọn): Danh sách các arguments mà prompt có thể chấp nhận
 - **name**: Tên của argument
 - **description** (tùy chọn): Mô tả về argument
 - **required** (tùy chọn): Xác định liệu argument có bắt buộc hay không

Ví dụ như đối với prompt `code_debug` ở trên, ta có định nghĩa về prompt đó trong MCP server như sau:

```
{
  "name": "code_debug",
  "description": "A tool to help debug code by providing the code and the error message.",
  "arguments": [
    {
      "name": "code",
      "description": "The code that has an error.",
      "required": true,
    },
    {
      "name": "error_message",
      "description": "The error message that was raised.",
      "required": true,
    },
  ],
}
```

```
    ],
}
```

Xây dựng Prompts cho Library MCP Server Chúng ta sẽ viết các prompts cho Library MCP Server với `@app.prompt()` decorator. Decorator `@prompt` có các arguments sau:

- `name`: Tên tùy chọn cho prompt (mặc định là tên của hàm)
- `title`: Tiêu đề hiển thị dễ hiểu dành cho con người (tùy chọn)
- `description`: Mô tả tùy chọn về chức năng hoặc mục đích của prompt. Nếu field này bị bỏ trống, phần docstring của hàm sẽ được sử dụng thay thế.

Tiếp theo, chúng ta implement các loại prompts sau:

- Static prompt: prompt không nhận bất kì tham số/biến nào. Trong MCP server của chúng ta, đó là prompt `suggest_random_book`.
- Dynamic prompt: prompt điều chỉnh theo arguments mà MCP client cung cấp. Bao gồm các prompts sau:
 - `suggest_book_title_by_abstract(abstract)`: nhận vào argument `abstract` để gọi ý tên sách dựa trên abstract.
 - `analyze_book(book, query)`: nhận vào hai arguments là `book` và `query` để phân tích sách dựa trên nội dung và câu hỏi của người dùng.

Tương tự như tools và resources, chúng ta cần implement hai hàm `list_prompts()` và `get_prompt(name, arguments)` để trả về danh sách các prompts và lấy một prompt cụ thể dựa trên tên prompt cần lấy cùng với các arguments tương ứng. Các hàm này sẽ trả về các đối tượng `Prompt` và `GetPromptResult` tương ứng với định nghĩa của MCP .

Đối với hàm `list_prompts()`, chúng ta dùng decorator `@server.list_prompts()` và implement như sau:

```
1 @server.list_prompts()
2 async def list_prompts() -> list[Prompt]:
3     return [
4         Prompt(
5             name="suggest_random_book",
6             description="Suggest a random book from the library. The suggestion should
7                 include the title, author, and a brief
8                 description.",
9         ),
10        Prompt(
11            name="suggest_book_title_by_abstract",
12            description="Suggest a memorable, descriptive title for a book based on the
13                 following abstract.",
14            arguments=[
15                PromptArgument(
16                    name="abstract",
17                    description="The abstract of the book.",
18                    required=True,
19                )
20            ],
21        ),
22    ],
```

```

18 ),
19 Prompt(
20     name="analyze_book",
21     description="Analyze a book based on its content and user query.",
22     arguments=[
23         PromptArgument(name="book", description="The book to analyze.", required=True
24             ),
25         PromptArgument(
26             name="query",
27             description="The query for analysis.",
28             required=True,
29         ),
30     ],
31 )

```

Chúng ta sẽ định nghĩa các prompts bao gồm các fields `name` là tên prompt, `description` chứa miêu tả về prompt cũng như cách sử dụng prompt, và `arguments` là các tham số của prompt (nếu có). Các arguments sẽ là một danh sách các object thuộc class `PromptArgument` với các fields `name` là tên của argument, `description` là miêu tả về argument, và `required` để xác định liệu argument đó có bắt buộc hay không.

Đối với hàm `get_prompt(name, arguments)` để trả về một prompt cụ thể, chúng ta dùng decorator `@server.get_prompt()` và implement như sau:

```

1 @server.get_prompt()
2 async def get_prompt(name: str, arguments: Dict[str, str] | None) -> GetPromptResult:
3     prompts = await list_prompts()
4     for prompt in prompts:
5         if prompt.name == name:
6             if arguments is None:
7                 arguments = {}
8             if prompt.arguments:
9                 for arg in prompt.arguments:
10                     if arg.name not in arguments and arg.required:
11                         raise ValueError(f"Missing required argument: {arg.name}")
12             break
13     else:
14         raise ValueError(f"Prompt '{name}' not found.")
15
16     if name == "suggest_random_book":
17         prompt_result = library.get_suggesting_random_book_prompt()
18         return GetPromptResult(
19             description=prompt.description,
20             messages=[
21                 PromptMessage(
22                     role="user",
23                     content=TextContent(type="text", text=prompt_result),
24                 )
25             ],
26         )
27     elif name == "suggest_book_title_by_abstract":
28         prompt_result = library.get_suggesting_book_title_by_abstract_prompt(**arguments)
29         return GetPromptResult(
30             description=prompt.description,

```

```

31     messages=[
32         PromptMessage(
33             role="user",
34             content=TextContent(type="text", text=prompt_result),
35         )
36     ],
37 )
38 elif name == "analyze_book":
39     book, query = arguments["book"], arguments["query"]
40     messages = library.get_analyzing_book_messages(book, query)
41     return GetPromptResult(
42         description=prompt.description,
43         messages=[
44             PromptMessage(
45                 role=m["role"],
46                 content=TextContent(type="text", text=m["content"]),
47             )
48             for m in messages
49         ],
50     )
51 else:
52     raise ValueError(f"Prompt '{name}' is not implemented.")

```

Hàm `get_prompt` sẽ nhận vào tên của prompt (`name`) và các arguments tương ứng (`arguments`). Hàm sẽ tìm kiếm trong danh sách các prompts đã được định nghĩa, kiểm tra xem tên prompt có tồn tại hay không, và nếu có thì kiểm tra các arguments có hợp lệ hay không. Nếu hợp lệ, hàm sẽ trả về một đối tượng `GetPromptResult` chứa prompt hoặc các messages để người dùng có thể sử dụng trong cuộc hội thoại với mô hình AI.

II.4.4. Tóm tắt các thành phần cơ bản của MCP Server

Thông tin tóm tắt về từng primitive được tóm tắt trong [Bảng 1](#).

Primitive	Điều khiển bởi	Mô tả	Ví dụ
Prompts	Người dùng kiểm soát	Mẫu tương tác được kích hoạt theo lựa chọn của người dùng	Lệnh gạch chéo, tùy chọn trong menu
Resources	Ứng dụng kiểm soát	Dữ liệu ngữ cảnh được đính kèm và quản lý bởi client	Nội dung file PDF, lịch sử git
Tools	Mô hình kiểm soát	Các hàm được mở ra cho LLM để thực hiện hành động	POST request đến external API, ghi file vào local

Bảng 1: Cấu trúc phân cấp điều khiển của các primitive

II.5. Các cơ Chế Truyền Tải trong MCP

Cơ chế truyền tải (transport) trong MCP xác định cách thức mà MCP client và MCP server trao đổi tin tức/thông điệp (các JSON-RPC messages) với nhau. Các cơ chế truyền tải này xử lý cách đóng gói, truyền tải và nhận thông điệp - đồng thời đảm bảo kết nối đáng tin cậy và

dễ tương tác giữa các hệ thống. Hiện tại, MCP hỗ trợ hai cơ chế truyền tải mặc định: stdio và Streamable HTTP. Ngoài ra, giao thức còn cho phép developer cài đặt các cơ chế truyền tải tùy chỉnh, mang lại sự linh hoạt trong việc tích hợp với các môi trường khác nhau.

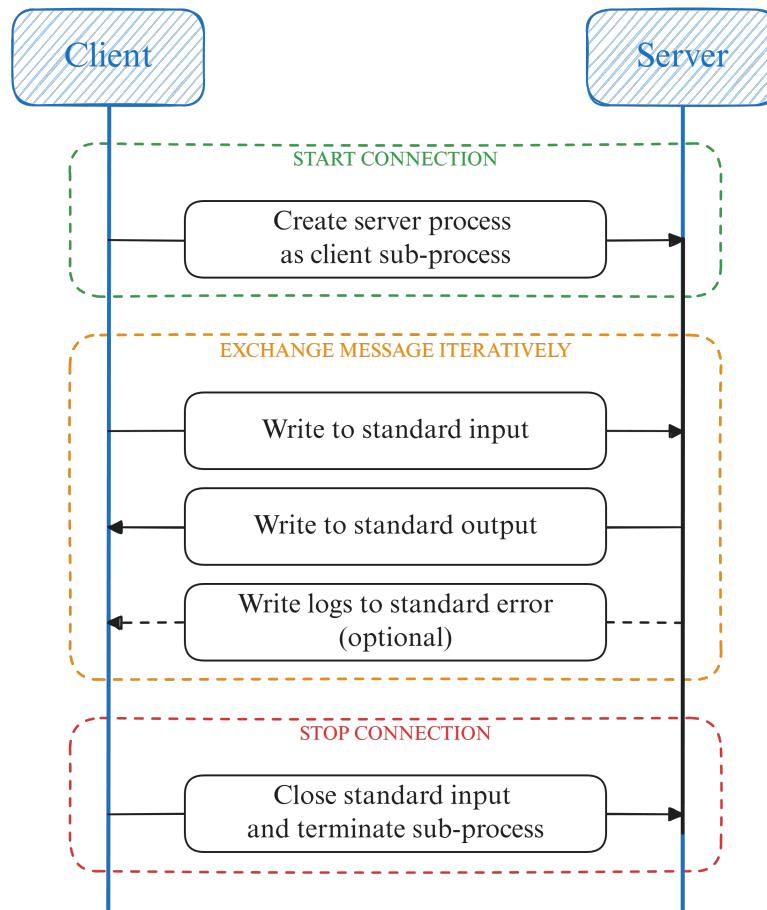
II.5.1. Standard Input/Output (stdio) Transport

Transport stdio cho phép giao tiếp thông qua các luồng input và output theo tiêu chuẩn. Chúng ta lựa chọn stdio transport cho các môi trường cục bộ như terminal, shell scripts, hoặc bất kỳ kịch bản huống mà MCP client và MCP server hoạt động trên cùng một máy.

Cơ chế hoạt động của stdio transport gồm những bước sau (minh họa trong [Hình 13](#)):

1. Client khởi chạy MCP server dưới dạng một tiến trình con (sub-process) của client process.
2. Các thông điệp được MCP client truyền qua standard input của MCP server và gửi đi từ MCP server qua standard output của MCP server. Quá trình này được lặp đi lặp lại trong suốt chu kì kết nối.
3. MCP server có thể ghi thông tin lỗi vào stderr, và MCP client có thể lựa chọn sử dụng log đó hoặc bỏ qua.
4. Khi kết thúc, MCP client đóng standard input và dừng tiến trình con.

Transport này phù hợp nhất cho các tích hợp đơn giản trong môi trường cục bộ, với thiết lập tối giản và hiệu suất cao mà không cần kết nối internet giữa MCP client và MCP server.



Hình 13: Stdio Transport Sequence Diagram

II.5.2. Streamable HTTP Transport

Transport Streamable HTTP được thiết kế cho các môi trường web hoặc nhiều client. Cơ chế này sử dụng HTTP POST để MCP client gửi thông điệp JSON-RPC tới MCP server, và tùy chọn sử dụng Server-Sent Events (SSE) để MCP server truyền thông điệp ngược lại theo kiểu streaming. Những đặc điểm nổi bật của cơ chế truyền tải này bao gồm:

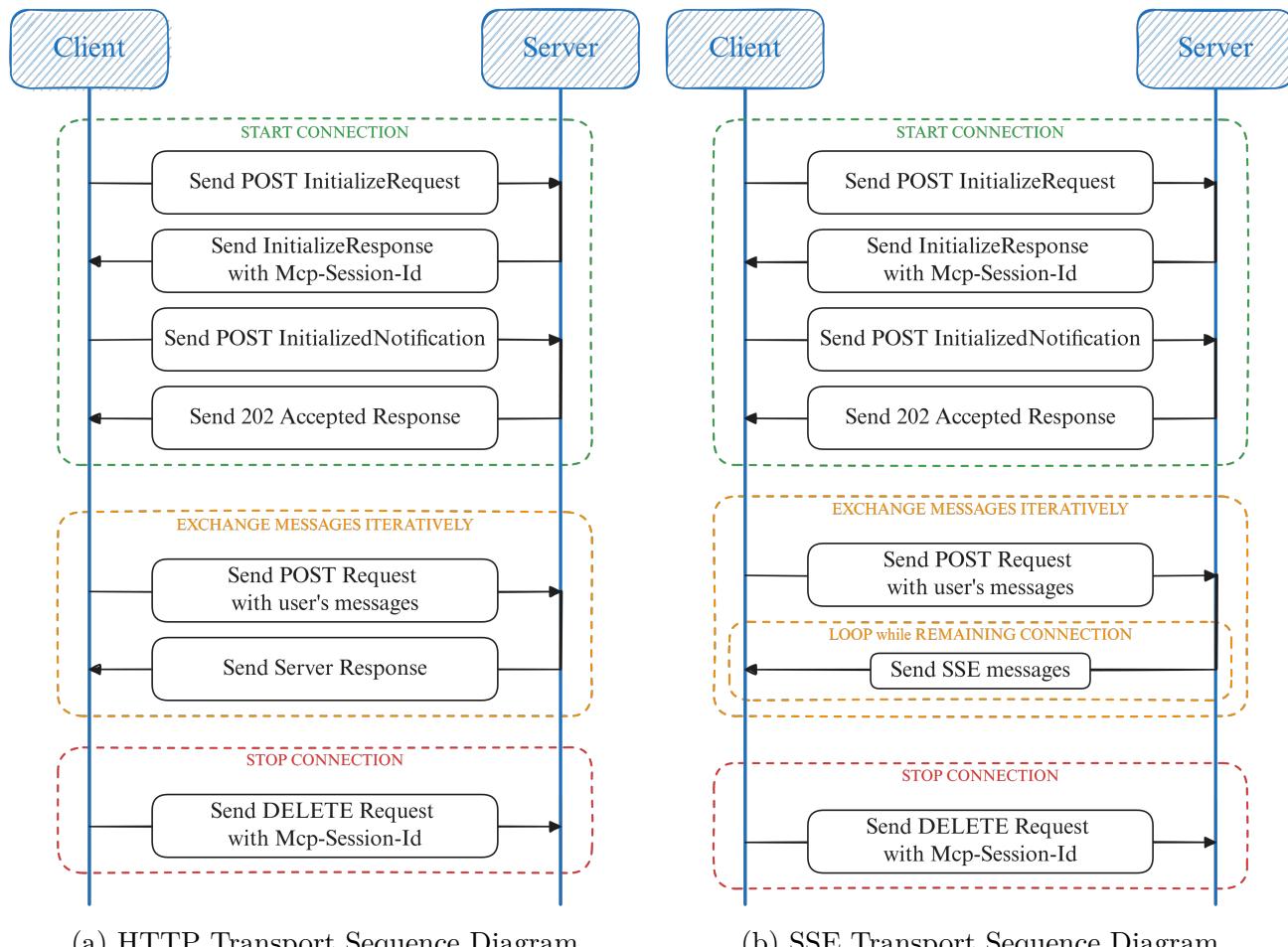
- **Session có chứa trạng thái (Stateful Session):** MCP client và MCP server có thể trao đổi ID phiên thông qua header `Mcp-Session-Id` tùy chỉnh nhằm duy trì ngữ cảnh xuyên suốt nhiều requests. Session bắt đầu khi khởi tạo và có thể được kết thúc rõ ràng qua method HTTP DELETE.
- **Streaming và khả năng tiếp tục khi bị gián đoạn:** MCP server có thể nhúng event ID trong phản hồi SSE để MCP client có thể tiếp tục các luồng bị gián đoạn bằng cách gửi `Last-Event-ID` trong các GET requests kế tiếp.
- **Xử lý thông điệp linh hoạt:** MCP client gửi JSON-RPC message qua POST request. MCP server có thể phản hồi bằng một JSON duy nhất hoặc khởi tạo luồng phản hồi liên tục thông qua SSE. Ngoài ra, MCP client cũng có thể mở các luồng SSE GET method để nhận các yêu cầu hoặc thông báo từ MCP server.
- **Yêu cầu về bảo mật:** Các quy tắc bao gồm kiểm tra header origin, giới hạn binding ở

localhost cho triển khai cục bộ, bắt buộc sử dụng TLS trong môi trường production, và quản lý session ID một cách an toàn.

Streamable HTTP Transport hỗ trợ tốt kiến trúc client-server, cho phép nhiều client hoạt động đồng thời tại Host, có thể phục hồi kết nối bị gián đoạn, và khả năng truyền thông điệp đa dạng hơn stdio transport.

Chu trình hoạt động của Streamable HTTP transport được liệt kê như sau (xem Hình 14a):

1. Khi bắt đầu kết nối, MCP client gửi POST request đến server để bắt đầu kết nối. Nếu MCP server chấp nhận kết nối, MCP server sẽ gửi về Mcp-Session-Id để phục vụ mục đích theo dõi và hỗ trợ phiên kết nối (reconnect, close, v.v.).
2. Trong quá trình sử dụng, MCP client sẽ gửi POST request chứa nội dung của người dùng hoặc là tool cần chạy, resource cần lấy, v.v. và MCP server sẽ xử lý nội dung và trả về trong một request tương tự. Đối với SSE transport (phiên bản cũ), response từ MCP server có thể được chia nhỏ thành nhiều messages và gửi đi lần lượt đến MCP client (xem Hình 14b).
3. Cuối cùng, để đóng kết nối, MCP client gửi DELETE request kèm theo Mcp-Session-Id trong header và server sẽ đóng kết nối với MCP client này.



Hình 14: Streamable HTTP Transport Sequence Diagram

II.5.3. SSE Transport

Trước đây, MCP hỗ trợ server-sent events (SSE) độc lập kết hợp với HTTP POST từ phía client, nhưng cơ chế này đã bị ngừng hỗ trợ kể từ ngày 5 tháng 11 năm 2024. Transport Streamable HTTP vẫn giữ khả năng hỗ trợ SSE nhưng theo một kiến trúc tiêu chuẩn và đầy đủ tính năng hơn. Các hệ thống MCP client và MCP server vẫn có thể duy trì tính tương thích với các phiên bản thư viện MCP mới thông qua các cơ chế hỗ trợ ngược (backward-compatibility).

II.5.4. Custom Transports

Bên cạnh hai cơ chế truyền tải stdio và Streamable HTTP, MCP không phụ thuộc vào transport, từ đó cho developer triển khai các lớp transport tùy biến theo nhu cầu. Điều này tạo điều kiện tích hợp trong các môi trường chuyên biệt - như message queues, framework điều khiển từ xa (remote procedure call frameworks - RPC), hoặc các kênh trên thiết bị di động. Các transport đều có thể được sử dụng với MCP, miễn là transport đó tuân thủ định dạng truyền tải thông điệp JSON-RPC của MCP, bao gồm quy tắc đóng gói thông điệp và đảm bảo vòng đời kết nối.

II.5.5. Implement MCP Client with Different Transport Mechanisms

Trong phần này, chúng ta sẽ implement các cơ chế truyền tải cho Library MCP Server vừa xây dựng với hai loại cơ chế Standard Input/Output và Streamable HTTP.

Standard Input/Output (stdio) Transport Trong đoạn code [II.4.](#), Library MCP Server đang sử dụng stdio transport (định nghĩa ở dòng 74-75) như sau:

Library MCP Server với stdio transport

```
1 options = server.create_initialization_options()
2 async with stdio_server() as (read_stream, write_stream):
3     await server.run(read_stream, write_stream, options)
```

Trong đó, các câu lệnh được giải thích như sau:

- `server.create_initialization_options()`: tạo ra một tập hợp các tùy chọn khởi tạo ban đầu cho MCP server. Các tùy chọn này có thể bao gồm thông tin về tool, resource, prompt, hoặc metadata,
- `async with stdio_server() as (read_stream, write_stream)`: Thiết lập một cơ chế truyền tải thông qua standard input/output với hai luồng `read_stream` để đọc các thông điệp từ MCP client gửi đến (ví dụ: các yêu cầu JSON-RPC) và `write_stream` để gửi thông điệp đến MCP client. Bên cạnh đó, thời nhở vào nhở vào cú pháp `async` của Python context manager, hệ thống tự động xử lý việc mở và đóng kết nối I/O một cách an toàn,
- `await server.run(read_stream, write_stream, options)` kích hoạt vòng lặp xử lý chính của MCP server. Keyword `await` đảm bảo coroutine này được thực thi bất đồng bộ, không chặn main process và không kết thúc cho đến khi MCP server bị tắt. Coroutine này đọc yêu cầu từ `read_stream`, xử lý thông điệp dựa trên cấu hình trong `options`, và gửi phản hồi qua `write_stream`.

Ở phía MCP client, chúng ta thiết lập stdio transport cho MCP client để mở một session như sau:

Client dùng stdio transport

```

1 async def test_mcp_server_with_stdio_transport():
2     async with stdio_client(server_params) as (read, write):
3         async with ClientSession(read, write) as session:
4             # Initialize the connection
5             await session.initialize()
6             ...

```

Trong đó, các câu lệnh

- `async with stdio_client(server_params) as (read, write):`: Khởi chạy một process phụ đại diện cho MCP server, thiết lập luồng đọc/ghi dựa trên các luồng standard input và standard output, đóng vai trò như kênh giao tiếp chính,
- `async with ClientSession(read, write) as session:`: bao bọc các luồng transport trong một lớp giao thức MCP ở tầng cao hơn, đồng thời cung cấp các method như `initialize()`, `list_tools()`, `call_tool()`, v.v.,
- `await session.initialize()` hoàn tất quá trình handshake với MCP server, bao gồm trao đổi metadata, các tool, prompt và resource hiện có của server, đồng thời chuẩn bị phiên làm việc để thực thi các lệnh liên quan đến tool và resource thông qua kênh kết nối đã thiết lập.

Sau khi đã khởi tạo kết nối thành công đến MCP server với lệnh `await session.initialize()`, chúng ta có thể khám phá năng lực của server thông qua các lệnh như `await session.list_tools()` và `await session.list_resources()`, đọc resource `await session.read_resource("resource://...")`, hay gọi tool để thực thi một task nào đó `result = await session.call_tool("tool_name", args=...)`

Streamable HTTP Transport Đối với cơ chế này, chúng ta thiết lập MCP server như sau:

Cài đặt Streamable HTTP MCP Server

```

1 session_manager = StreamableHTTPSessionManager(
2     app=server,
3     event_store=None,
4     json_response=(
5         True if transport == "sse" else False
6     ), # Use JSON response for SSE, otherwise standard HTTP
7     stateless=True,
8 )
9
10 # Handle HTTP requests with the session manager
11 async def handle_streamable_http(scope: Scope, receive: Receive, send: Send) -> None:
12     await session_manager.handle_request(scope, receive, send)
13
14 @contextlib.asynccontextmanager

```

```

15|     async def lifespan(app: Starlette) -> AsyncIterator[None]:
16|         async with session_manager.run():
17|             logger.info("Application started with StreamableHTTP session manager!")
18|             try:
19|                 yield
20|             finally:
21|                 logger.info("Application shutting down...")
22|
23| # Create an ASGI application using the transport
24| starlette_app = Starlette(
25|     debug=True,
26|     routes=[
27|         Mount("/mcp", app=handle_streamable_http),
28|     ],
29|     lifespan=lifespan,
30| )
31|
32| uvicorn.run(starlette_app, host="127.0.0.1", port=port, log_level=log_level.lower())

```

Đoạn code này được giải thích như sau:

- Khởi tạo Session Manager với class `StreamableHTTPSessionManager` và nhận vào các arguments bao gồm:
 - `app=server`: MCP server chúng ta đã gắn các tools, resources và promtps,
 - `event_store=None`: tắt tính năng lưu event để hỗ trợ kết nối lại (resume). MCP client không thể tiếp tục session nếu bị disconnect,
 - `json_response`: Khi `transport=="http"`, trả về JSON response, và khi `transport == "sse"` thì trả về theo dạng streaming,
 - `stateless=True`: Mỗi request HTTP POST là một session riêng biệt, không ràng buộc về trạng thái hay session qua các kết nối.
- Tạo handler xử lý request đến ASGI¹ bằng function `handle_streamable_http(scope, receive, send)`: Function này được mount vào ASGI server (Starlette/uvicorn). Mọi request đến /mcp sẽ được chuyển qua `handle_request()`, nơi session manager sẽ parse, tạo session tạm thời mới, gọi tool-method và trả kết quả,
- Quản lý lifecycle của ứng dụng với hàm `lifespan(app)`: Trong hàm này, câu lệnh `session_manager.run()` mở một task group nội bộ để quản lý nhiều kết nối song song. Khi ứng dụng được khởi động, Starlette gọi hàm `lifespan(app)`, nhảy vào context tạo bởi `session_manager.run()` và session manager sẵn sàng nhận nhiều request cùng lúc. Khi shutdown, hệ thống thoát khỏi context và thông báo dừng server,
- Cấu hình ứng dụng Starlette với class `Starlette`: Khởi tạo ứng dụng ASGI và gắn server

¹ ASGI (Asynchronous Server Gateway Interface) là phiên bản hiện đại hơn của WSGI (Web Server Gateway Interface). Trong khi WSGI chỉ xử lý một yêu cầu web tại một thời điểm và phải chờ hoàn tất trước khi xử lý tiếp, thì ASGI có khả năng xử lý nhiều yêu cầu đồng thời mà không cần chờ đợi - giống như việc chúng ta có thể vừa trò chuyện vừa lướt web một cách mượt mà trong cùng một phiên làm việc. Framework Flask chỉ hỗ trợ WSGI, trong khi các Python web framework mới nhất hỗ trợ ASGI như FastAPI, Quart, Django (>=4), v.v.

của chúng ta vào route /mcp thông qua mount route /mcp và handler `handle_streamable_http`,

5. Khởi chạy server với Uvicorn: Bắt đầu chạy server ASGI của chúng ta ở local với method `uvicorn.run` và các tham số như `host`, `port`, v.v.

Về phía MCP client, chúng ta cài đặt để tạo kết nối tới MCP server bằng Streamable HTTP transport như sau:

Streamable HTTP MCP Client

```

1  async with streamablehttp_client(server_url) as (read, write, get_session_id_callback):
2      async with ClientSession(read, write) as session:
3          await session.initialize()
4          ...

```

Khác với MCP client dùng stdio transport ở [II.5.5.](#), câu lệnh `async with streamable_http_client(server_url) as (read, write, get_session_id_callback):`: tạo một kết nối HTTP đến MCP server tại `server_url`. Ngoài ra, `streamablehttp_client` sử dụng phương thức truyền tải hoạt động thông qua HTTP bằng gửi các thông điệp JSON-RPC bằng phương thức POST và nhận phản hồi từ MCP server (kể cả các streaming events). Ngoài ra, hàm `streamablehttp_client` còn cung cấp một callback `get_session_id_callback`, cho phép MCP client truy xuất session ID nhằm hỗ trợ quá trình reconnect khi bị mất kết nối.

II.6. Chạy Library MCP Server và thiết kế MCP Client để kiểm tra

Trong phần này, chúng ta sẽ tổng hợp lại toàn bộ đoạn code của Library MCP Server, MCP client và cách chạy các đoạn scripts.

II.6.1. Library MCP Server Full Implementation

Đầu tiên, ta tạo một folder rỗng và đặt tên liên quan đến project (vd: `library_mcp_server_and_client`). Sau đó, ta tạo file `server.py` với nội dung ở link [này](#).

Trong implementation của Library MCP Server, bên cạnh framework `mcp`, chúng ta dùng thêm thư viện `click` để thêm arguments trong quá trình chạy script trên terminal. Chúng ta có các arguments sau:

- `--log-level`: Cho phép người dùng chỉ định mức độ log của ứng dụng. Các giá trị hợp lệ là DEBUG, INFO, WARNING, ERROR, CRITICAL. Mặc định là INFO. Argument này giúp kiểm soát thông tin in ra terminal hiển thị khi server đang chạy.
- `--transport`: Tuỳ chọn để chọn loại transport giữa MCP server và MCP client. Các giá trị hợp lệ là http, stdio, sse. Mặc định là http. Argument này quy định cách thức trao đổi dữ liệu giữa MCP client và MCP server.
- `--port`: Cho phép người dùng chỉ định cổng (port) mà HTTP MCP server sẽ triển khai trên đó. Giá trị port mặc định là 8000 và các giá trị người dùng đặt phải là một số nguyên.

Argument này rất quan trọng khi triển khai server trên các môi trường khác nhau hoặc tránh xung đột cổng.

II.6.2. MCP Client Implementation

Trong folder vừa tạo, ta tạo tiếp một file `client.py` có nội dung đặt trong link [này](#).

File `client.py` có các thành phần chính sau:

- Các hàm hỗ trợ như `display_tools()`, `display_resources()`, `display_prompts()`, `display_resource_templates()` dùng để hiển thị các thành phần chính của MCP server (tools, resources, prompts).
- Hàm kiểm thử chính trong file `test_book_library_management_mcp_server(session)` thực hiện tuần tự các tác vụ kiểm thử cơ bản sau khi MCP client đã kết nối tới MCP server với các chức năng sau
 - Liệt kê các tool, resources, resource templates, prompts có sẵn (thể hiện rõ kiến trúc 3-primitives của MCP server đã mô tả ở các phần trên).
 - Thực hiện lần lượt các thao tác để tương tác với MCP server và hệ thống quản lý thư viện như sau:
 1. Gọi tool `get_num_books` để lấy số sách hiện có.
 2. Truy xuất toàn bộ resource `books://all` và in ra danh sách sách.
 3. Thêm một quyển sách mới thông qua tool `add_book` (tạo dữ liệu ngẫu nhiên để kiểm thử khả năng hoạt động thực tế).
 4. Kiểm tra số lượng sách sau khi thêm.
 5. Truy xuất quyển sách vừa thêm bằng index và bằng ISBN qua resource template.
 6. Xóa quyển sách vừa thêm bằng tool `remove_book`.
 7. Kiểm tra số lượng sách sau khi xóa, đảm bảo tính nhất quán dữ liệu.
 8. Lấy nội dung các prompt (bao gồm prompt static, dynamic) và in ra kết quả, giúp xác thực tính năng prompts đã trình bày trong MCP .
- Tích hợp và test từng loại transport:
 - `test_mcp_server_with_stdio_transport` khởi động MCP server sử dụng standard input/output transport, kết nối với client và test bằng các hàm trên.
 - `test_mcp_server_with_http_transport` kết nối tới MCP server qua HTTP (hoặc SSE), thử nghiệm toàn bộ workflow thông qua API chuẩn hoá theo MCP (JSON-RPC).

Luồng hoạt động tổng thể của MCP client chúng ta vừa thiết kế được tóm tắt như sau:

1. Lựa chọn transport phù hợp và kết nối tới MCP server (theo đúng kiến trúc client-server chuẩn của MCP).
2. Thực hiện handshake/initialize session với server.
3. Truy vấn metadata về server (tools, resources, resource templates, prompts).

4. Thực thi tuần tự các tác vụ: đọc/thêm/xoá sách, truy xuất resource động/tĩnh, lấy prompts.
5. In ra kết quả chi tiết từng bước để kiểm tra hoạt động và minh họa cho kiến trúc primitives của MCP .

II.6.3. Chạy MCP Server và Client với stdio Transport

Đối với stdio transport, chúng ta không cần chạy server. Thay vào đó, chúng ta đưa đường dẫn của file `server.py` đến cho client. Sau đó, chúng ta chạy file `client.py` từ terminal với lệnh:

```
python ./client.py --transport stdio
```

Kết quả nhận được là:

Kết quả chạy file `client.py` để test Library MCP Server

```
*** Available Tools ***
Tool: add_book
    Add a book to the library
Tool: remove_book
    Remove a book by its ISBN
Tool: get_num_books
    Get the total number of books

*** Available Resources ***
Resource: All Books (books://all)

*** Available Resource Templates ***
Resource Template: Book by Index (books://index/{index})
    Get a book by its index in the library
Resource Template: Book by ISBN (books://isbn/{isbn})
    Get a book by its ISBN

*** Available Prompts ***
Prompt: suggest_random_book (suggest_random_book)
    Suggest a random book from the library. The suggestion should include
    the title, author, and a brief description.
Prompt: suggest_book_title_by_abstract (suggest_book_title_by_abstract)
    Suggest a memorable, descriptive title for a book based on the
    following abstract.
Prompt: analyze_book (analyze_book)
    Analyze a book based on its content and user query.

#####
*** Calling Tool: get_num_books ***
Results=2

*** Get all books ***
All Books:
[
    {
        "title": "Atomic Habits",

```

```
        "author": "James Clear",
        "isbn": "9780735211292",
        "tags": [
            "self-help",
            "psychology",
            "productivity"
        ]
    },
{
    "title": "The Pragmatic Programmer",
    "author": "Andrew Hunt, David Thomas",
    "isbn": "9780201616224",
    "tags": [
        "software engineering",
        "programming",
        "career"
    ]
}
]

*** Adding a new book ***
Response from add_book: Book 'Random Book Name 979' by Random Author 382
added to the library.

*** Number of Books in Library after addition ***
Number of Books in Library: 3

*** Retrieve the added book by index ***
Retrieved Book:
{
    "title": "Random Book Name 979",
    "author": "Random Author 382",
    "isbn": "9787352249742",
    "tags": [
        "biography",
        "history",
        "poetry"
    ]
}

*** Retrieve the added book by ISBN ***
Retrieved Book:
{
    "title": "Random Book Name 979",
    "author": "Random Author 382",
    "isbn": "9787352249742",
    "tags": [
        "biography",
        "history",
        "poetry"
    ]
}

*** Removing the added book ***
```

```

Response from remove_book: Book with ISBN '9787352249742' removed from
the library.

*** Number of Books in Library after removal ***
Number of Books in Library: 2

#####
*** Get suggest_random_book prompt ***
Prompt Response: [PromptMessage(role='user', content=TextContent(type='
text', text='Suggest a random book from the library. The suggestion
should include the title, author, and a brief description.', annotations=None, meta=None))]

*** Get suggest_book_title_by_abstract prompt ***
Prompt Response: [PromptMessage(role='user', content=TextContent(type='
text', text='Suggest a memorable, descriptive title for a book based
on the following abstract: A book about the wonders of the universe
.', annotations=None, meta=None))]

*** Get analyze_book prompt ***
Prompt Response: [
{
    "role": "user",
    "content": "This is the book I want to analyze: \"{}\"": "
    "\\\"Random Book Name 979\\\"", \\"author\\\": \\"Random Author
    382\\\"", \\"isbn\\\": \\"9787352249742\\\"", \\"tags\\\": [\\""
    biography\\\", \\"history\\\", \\"poetry\\\"]}\\"
},
{
    "role": "assistant",
    "content": "Sure! Let's analyze this book together. What would you
    like to know?"
},
{
    "role": "user",
    "content": "What is the main theme of this book?"
}
]
#####

Test completed successfully!

```

Phân tích kết quả trả về, ta thấy được:

- Chúng ta thấy MCP client nhận được ba tools: add_book, remove_book, và get_num_books. Resource mà MCP client nhận được là All Books (books://all) và hai resource templates là Book by Index (books://index/index) và Book by ISBN (books://isbn/isbn). Cuối cùng, chúng ta có ba prompts là suggest_random_book, suggest_book_title_by_abstract, và analyze_book.

- Trong quá trình test các tools, MCP client đã thực hiện các thao tác như lấy số lượng sách, thêm sách mới, truy xuất sách theo index và ISBN, xóa sách, và cuối cùng là lấy nội dung của các prompts. Kết quả trả về cho thấy các thao tác đều thành công và dữ liệu được xử lý chính xác.
- Việc truy xuất các prompts cũng cho thấy tính năng này hoạt động tốt, với các prompt được trả về dưới dạng danh sách các tin nhắn (messages) với vai trò người dùng và trợ lý.

II.6.4. Chạy MCP Server và Client với Streamable HTTP Transport

Đối với giao thức này, chúng ta phải chạy MCP server trước với lệnh sau:

```
python ./server.py --transport http --port 8000
```

Sau đó, chúng ta chạy file client.py với lệnh:

```
python ./client.py --transport http
```

Kết quả trả về sẽ tương tự như khi chạy với stdio transport, nhưng thay vì sử dụng I/O của subprocess, nó sẽ sử dụng HTTP để giao tiếp với MCP server.

Toàn bộ source code của Library MCP Server được lưu trữ tại [đây](#).

II.6.5. MCP Inspector

MCP Inspector là một công cụ tương tác dành cho nhà phát triển, hỗ trợ kiểm thử và gỡ lỗi các MCP server.

Để khởi động MCP Inspector, ta cần cài trước [Node.js](#) (≥ 22.7.5). Sau đó, trong folder của server đã tạo ở trên (folder `library_mcp_server_and_client`), chúng ta chạy lệnh sau trong terminal:

```
npx @modelcontextprotocol/inspector
```

Output nhận được trong terminal là:

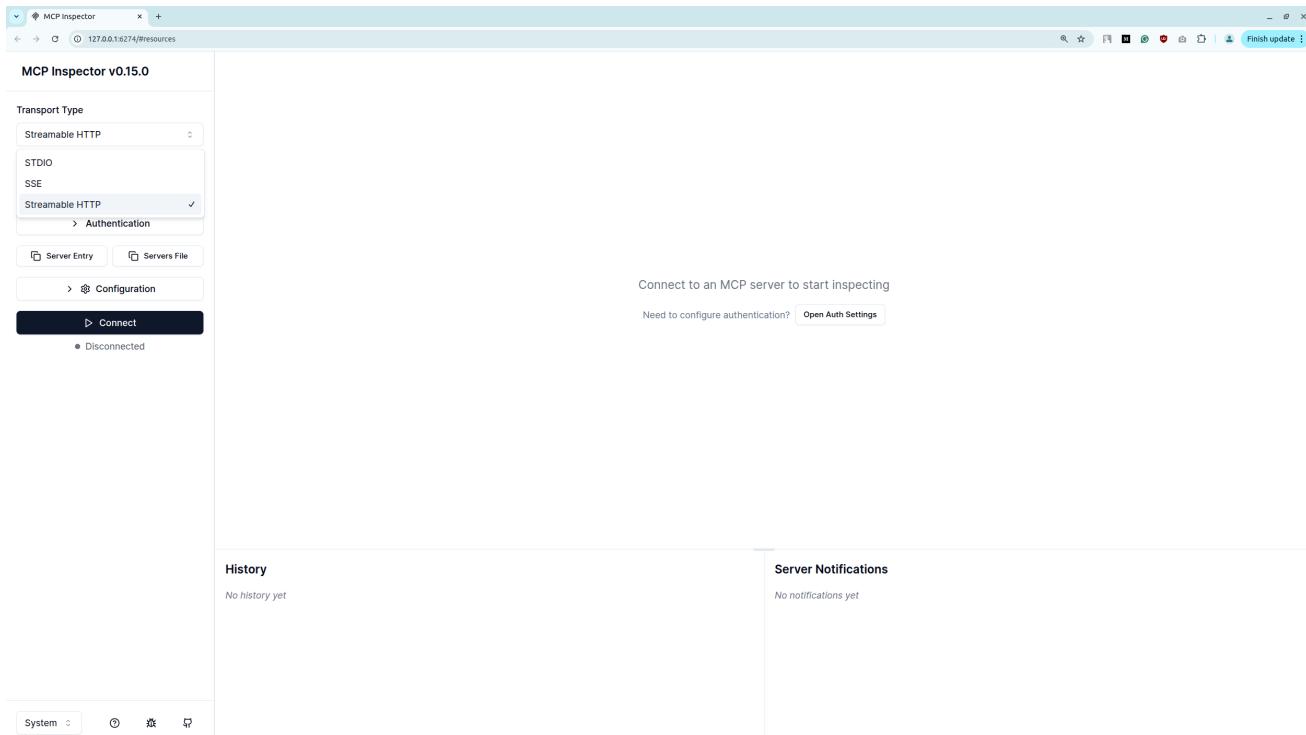
Kết quả nhận được trong terminal

```
Starting MCP inspector...
Proxy server listening on 127.0.0.1:6277
Session token: ...
Use this token to authenticate requests or set DANGEROUSLY OMIT AUTH=true to
disable auth
```

Open inspector with token pre-filled:

http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=...

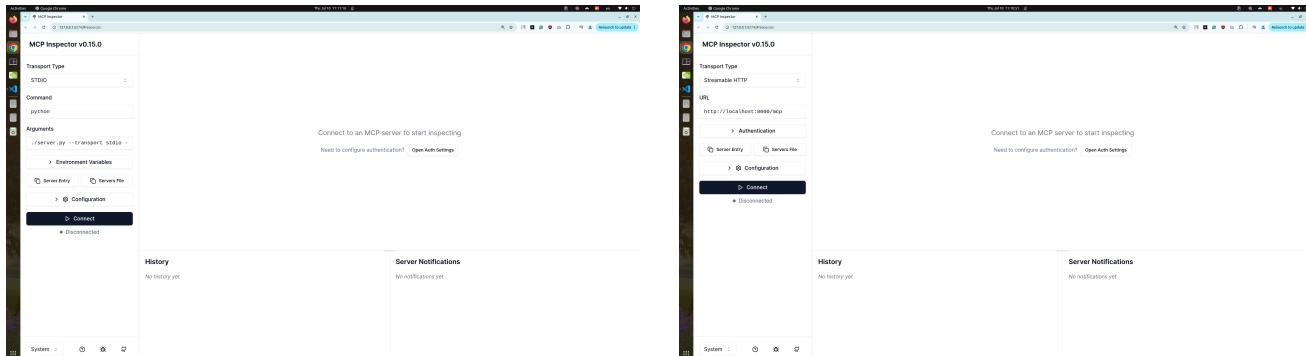
Chúng ta bật trình duyệt và truy cập vào URL có dạng: http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=... trong output của terminal để sử dụng MCP Inspector. Giao diện của MCP Inspector giống với Hình 15.



Hình 15: MCP Inspector Homepage

Đầu tiên, chúng ta sẽ kết nối với MCP server bằng sidebar ở bên trái (xem Hình 16). Đối với từng giá trị trong mục Transport Type, ta điền như sau:

- Nếu chúng ta chọn STDIO thì điền các field như sau:
 - Command: python,
 - Arguments: ./server.py --transport stdio --log-level error
- Nếu chúng ta chọn Streamable HTTP thì chúng ta sẽ điền field URL là http://localhost:8000/mcp. và chúng ta ấn Connect để kết nối với server.



(a) STDIO

(b) Streamable HTTP

Hình 16: Filling Required Fields to Connect to MCP

Với MCP Inspector, ta có thể xem các tài nguyên mà MCP server cung cấp, bao gồm Tools,

Resources và Prompts như Hình 17.

The screenshot shows the MCP server Inspector interface with the 'Tools' tab selected. The left panel, titled 'Tools', lists three commands: 'add_book', 'remove_book', and 'get_num_books'. Each command has a brief description below it. The right panel, titled 'Select a tool', contains a placeholder text: 'Select a tool from the list to view its details and run it'.

Tool	Description
<code>add_book</code>	Add a book to the library
<code>remove_book</code>	Remove a book by its ISBN
<code>get_num_books</code>	Get the total number of books

(a) Tools

The screenshot shows the MCP server Inspector interface with the 'Resources' tab selected. The left panel, titled 'Resources', lists 'List Resources' and 'Clear'. Below these is a list item 'all_books' with a dropdown arrow. The middle panel, titled 'Resource Templates', lists 'List Templates' and 'Clear', followed by two template items: 'book_by_index' and 'book_by_isbn', each with a dropdown arrow. The right panel, titled 'Select a resource or template', contains a placeholder text: 'Select a resource or template from the list to view its contents'.

Resource	Description
<code>all_books</code>	Syntax: <code>all_books</code>

(b) Resources

The screenshot shows the MCP server Inspector interface with the 'Prompts' tab selected. The left panel, titled 'Prompts', lists 'List Prompts' and 'Clear'. Below these are three prompt entries: 'suggest_random_book', 'suggest_book_title_by_abstract', and 'analyze_book'. Each entry has a detailed description. The right panel, titled 'Select a prompt', contains a placeholder text: 'Select a prompt from the list to view and use it'.

Prompt	Description
<code>suggest_random_book</code>	Suggest a random book from the library. The suggestion should include the title, author, and a brief description.
<code>suggest_book_title_by_abstract</code>	Suggest a memorable, descriptive title for a book based on the following abstract.
<code>analyze_book</code>	Analyze a book based on its content and user query.

(c) Promtps

Hình 17: Xem các core components của MCP server với Inspector

III. Cài đặt Personal Database MCP Server

III.1. Tạo thư mục project và cài thư viện

Chúng ta thực hiện những bước sau để tạo thư mục project và cài ứng dụng uv để quản lý thư viện của Python:

- Cài uv

Cài uv trên Windows

```
powershell -ExecutionPolicy Bypass -c "irm https://astral.sh/uv/install.ps1 | iex"
```

Cài uv trên macOS

```
curl -LsSf https://astral.sh/uv/install.sh | sh
```

- Tạo project folder và di chuyển vào thư mục project:

Tạo project folder với uv

```
uv init Personal-Database-MCP-Server
cd Personal-Database-MCP-Server/
```

Một thư mục tên `Personal-Database-MCP-Server` được tạo ra bằng uv và chúng ta sẽ di chuyển vào thư mục này để implement MCP server của chúng ta.

- Trong folder của project , chúng ta tạo thêm các folder và file theo cấu trúc như sau (nếu file nào có sẵn do được tạo bởi uv thì ta bỏ qua):

```
personal_database_mcp_server/
|-- documents/
|-- .python-version
|-- create_vector_database.py
|-- prepare_documents.py
|-- pyproject.toml
|-- README.md
|-- retriever.py
|-- server.py
|-- uv.lock
```

- Cài Python cho riêng project này với uv và kích hoạt Python:

Cài python riêng cho project trên Windows bằng uv

```
uv venv
.venv\Scripts\activate
```

Cài python riêng cho project trên macOS bằng uv

```
uv venv
source .venv/bin/activate
```

5. Cài thư viện:

Cài thư viện với uv trên Windows

```
uv add click mcp[cli] httpx langchain-community langchain openai
duckduckgo-search ddgs sentence-transformers transformers qdrant-
client
```

Cài thư viện với uv trên macOS

```
uv add click "mcp[cli]" httpx langchain-community langchain openai
duckduckgo-search ddgs sentence-transformers transformers qdrant-
client
```

III.2. Chuẩn bị dữ liệu và văn bản

Trong file `prepare_documents.py`, chúng ta sẽ tải về các bộ dữ liệu từ HuggingFace Datasets và lưu vào folder `documents`.

Để thuận tiện cho việc chuẩn bị data, chúng ta sẽ sử dụng các bộ dữ liệu có sẵn từ HuggingFace Datasets. Các bộ dữ liệu này chứa các định nghĩa về keywords của từng ngành học, giúp chúng ta xây dựng một cơ sở dữ liệu cá nhân (Personal Database) phục vụ cho việc tìm kiếm và truy vấn thông tin. Danh sách các bộ dữ liệu này được lưu trong biến `dataset_ids` như sau:

Danh sách tên các datasets dùng trong project này

```
1 dataset_ids = [
2     "burgerbee/pedagogy_textbook",
3     "burgerbee/pedagogy_wiki",
4     "burgerbee/psychology_textbook",
5     "burgerbee/psychology_wiki",
6     "burgerbee/psychiatry_textbook",
7     "burgerbee/psychiatry_wiki",
8     "burgerbee/art_and_culture_textbook",
9     "burgerbee/art_and_culture_wiki",
10    "burgerbee/medicine_textbook",
```

```

11 "burgerbee/medicine_wiki",
12 "burgerbee/chemistry_textbook",
13 "burgerbee/chemistry_wiki",
14 "burgerbee/social_studies_textbook",
15 "burgerbee/social_studies_wiki",
16 "burgerbee/religion_textbook",
17 "burgerbee/religion_wiki",
18 "burgerbee/science_studies_textbook",
19 "burgerbee/science_studies_wiki",
20 "burgerbee/history_textbook",
21 "burgerbee/history_wiki",
22 "burgerbee/philosophy_textbook",
23 "burgerbee/philosophy_wiki",
24 "burgerbee/biology_textbook",
25 "burgerbee/biology_wiki",
26 "burgerbee/physics_textbook",
27 "burgerbee/physics_wiki",
28 ]

```

Đây là những bộ dữ liệu chứa các định nghĩa về keywords của từng ngành.

Tiếp theo, chúng ta download dữ liệu trên về và lưu vào local trong folder documents như sau:

Tải các datasets về local storage

```

1 from datasets import load_dataset
2 import os
3 import json
4 from datasets.utils.logging import set_verbosity_error
5 import tqdm
6
7 set_verbosity_error()
8
9 CACHE_DIR = "./cache"
10 DOCUMENT_DIR = "./documents"
11
12 dataset_ids = [
13     ... # dataset ids here (similar to dataset_ids mentioned in the previous code block
14     )
15 ]
16
17 for dataset_id in tqdm.tqdm(dataset_ids, desc="Downloading datasets"):
18     print(f"Downloading {dataset_id}...")
19     dataset = load_dataset(dataset_id, cache_dir=CACHE_DIR)
20     print(f"Downloaded {dataset_id} with {len(dataset)} splits.")
21
22     dataset_dir = os.path.join(DOCUMENT_DIR, dataset_id.split("/")[1])
23     os.makedirs(dataset_dir, exist_ok=True)
24
25     for split in dataset:
26         print(f" - {split}: {len(dataset[split])} examples")
27         split_data = len(dataset[split])

```

```

27
28     for i, sample in enumerate(dataset[split]):
29         title, text = sample["title"], sample["text"]
30         source = f"https://hf.co/datasets/{dataset_id}"
31         sample["source"] = source
32         sample["split"] = split
33         sample["id"] = f"{dataset_id}-{split}-{i}"
34
35         doc_path = os.path.join(dataset_dir, f"{title.replace('/', '_')}{i}.json")
36         with open(doc_path, "w", encoding="utf-8") as f:
37             json.dump(sample, f, ensure_ascii=False, indent=4)
38
39         print(f"Saved {split_data} samples to {dataset_dir}")
40         print(f"Finished processing {dataset_id}.\n")
41 print("All datasets processed successfully.")

```

Dữ liệu tải về và được lưu vào máy trong các folder tương ứng với dataset name và split của nó. Ví dụ, nếu chúng ta tải về bộ dữ liệu burgerbee/art_and_culture_textbook, các documents sẽ được lưu trong folder documents/art_and_culture_textbook với các file json tương ứng với từng sample trong bộ dữ liệu. Cấu trúc của thư mục documents có dạng như sau:

```

1 documents/
2 |-- art_and_culture_textbook/
3 |   |-- 21st century skills_18.json
4 |   |-- Alternative fashion_40.json
5 |   |-- ...
6 |
7 |-- history_textbook/
8 |   |-- Ancient civilizations_01.json
9 |   |-- World Wars_52.json
10 |   |-- ...
11 |
12 |-- science_textbook/
13 |   |-- Physics fundamentals_14.json
14 |   |-- Chemistry basics_22.json
15 |   |-- ...
16 |
17 |-- ...

```

III.3. Xây dựng Vector Store

Chúng ta sẽ chuyển đổi các documents đã tải về thành embedding vectors tương ứng, phục vụ cho việc lưu trữ và tìm kiếm sau này. Việc chuyển đổi này sẽ sử dụng một mô hình embedding đã được huấn luyện trước, giúp chúng ta tạo ra các vector đại diện cho nội dung của từng document.

Giai đoạn này sẽ được implement ở file theo các bước sau:

1. Với các documents đã tải xuống và lưu ở folder documents, chúng ta tìm tất cả vị trí của documents với hàm `discover_and_get_all_files`:

Hàm discover_and_get_all_files

```

1 def discover_and_get_all_files(root_dir, allowed_extensions=None, recursive=False):
2     :
3     if allowed_extensions is None:
4         allowed_extensions = ['.json', '.txt', '.md']
5
6     all_files = []
7     for item_name in os.listdir(root_dir):
8         item_path = os.path.join(root_dir, item_name)
9         if os.path.isfile(item_path) and any(item_path.endswith(ext) for ext in
10                                         allowed_extensions):
11             all_files.append(item_path)
12         elif os.path.isdir(item_path) and recursive:
13             all_files.extend(discover_and_get_all_files(item_path,
14                                               allowed_extensions, recursive))
15
16     return all_files
17
18
19 all_document_paths = discover_and_get_all_files(DOCUMENT_DIR, allowed_extensions
20                                                 =['.json', '.txt', '.md'], recursive=True
21 )
22
23 print(f"Found {len(all_document_paths)} documents in {DOCUMENT_DIR}.")
24 ### OUTPUT: Found 24954 documents in ./documents.

```

2. Chúng ta implements các hàm đọc documents từ file lưu trên ổ cứng như sau:

Các hàm để đọc file lưu trên local storage

```

1 def load_json_file(file_path):
2     with open(file_path, 'r', encoding='utf-8') as f:
3         return json.load(f)
4
5 def load_txt_file(file_path):
6     with open(file_path, 'r', encoding='utf-8') as f:
7         return f.read()
8
9 def load_md_file(file_path):
10    with open(file_path, 'r', encoding='utf-8') as f:
11        return f.read()

```

Các hàm này sẽ giúp chúng ta đọc các file json, txt và md từ ổ cứng. Hàm `load_json_file` sẽ đọc file json và trả về nội dung của nó dưới dạng dictionary. Tương tự, hàm `load_txt_file` và `load_md_file` sẽ đọc file txt và md tương ứng và trả về nội dung văn bản của chúng. Những hàm này sẽ được sử dụng trong MCP server của chúng ta khi chúng ta load một văn bản nào đó để trả về dưới dạng resource.

3. Khi có được, vị trí của chúng lưu tại `all_document_paths`, chúng ta sẽ load tất cả documents từ các file đã lưu trong folder `documents` và lưu chúng vào biến `documents`:

Load tất cả các documents lên memory

```

1 documents = []
2 with tqdm(total=len(all_document_paths), desc="Loading documents") as pbar:
3     for doc_path in all_document_paths:
4         if doc_path.endswith('.json'):
5             doc = load_json_file(doc_path)
6         elif doc_path.endswith('.txt'):
7             doc = {"text": load_txt_file(doc_path)}
8         elif doc_path.endswith('.md'):
9             doc = {"text": load_md_file(doc_path)}
10        else:
11            continue
12
13        # Add metadata
14        doc["path"] = doc_path
15        documents.append(doc)
16        pbar.update(1)

```

Mỗi document sẽ là một dictionary chứa nội dung văn bản của document với field `text` và các metadata khác như `path`, `source`, `split`, `id` (nếu có).

- Để chuyển đổi documents thành embedding, chúng ta dùng embedding model [Alibaba-NLP/gte-multilingual-base](#) và load model bằng thư viện `SentenceTransformers` như sau:

Load embedding model

```

1 embedding_model_id = "Alibaba-NLP/gte-multilingual-base"
2 embedding_model = SentenceTransformer(
3     embedding_model_id,
4     trust_remote_code=True,
5     cache_folder=".cache"
6 )

```

và convert các documents đó về embeddings:

Chuyển các documents đã load về embedding

```

1 document_embeddings = embedding_model.encode(
2     [doc["text"] for doc in documents],
3     show_progress_bar=True,
4     convert_to_tensor=True,
5     normalize_embeddings=True,
6     batch_size=16,
7 )

```

- Để lưu trữ embedding của documents, chúng ta dùng [Qdrant Vector Database](#) cùng với thư viện [qdrant-client](#). Qdrant là một vector similarity search engine, cung cấp dịch vụ sẵn sàng cho production với API tiện lợi hỗ trợ việc lưu trữ, tìm kiếm và quản lý các points (vectors) với payload đi kèm. Chúng ta có thể coi payload là các thông tin bổ sung đi kèm

với vector, hỗ trợ việc tìm kiếm, truy vấn và lưu trữ thông tin. Đầu tiên, chúng ta tạo Qdrant Database chạy ở local bằng một trong hai cách sau:

- (a) Dữ liệu được lưu trực tiếp vào folder trên ổ cứng:

Cách này phù hợp cho việc xây dựng ứng dụng trong quá trình phát triển và kiểm thử. Chúng ta trực tiếp tạo `qdrant_client` trong Python như sau:

Tạo Qdrant client để kết nối tới Qdrant database ở local

```
1 client = QdrantClient(path="../qdrant_database")
```

- (b) Dữ liệu được quản lí trong Docker Container:

Cách này phù hợp khi ứng dụng đã vào giai đoạn triển khai (production). Để chạy Qdrant Database dưới dạng một Docker container, chúng ta chạy lệnh sau trong terminal:

Chạy Qdrant database với Docker

```
docker run -p 6333:6333 -p 6334:6334 \
-v "$(pwd)/qdrant_storage:/qdrant/storage:z" \
qdrant/qdrant
```

Sau đó, chúng ta kết nối và tương tác với Qdrant container thông qua `client` bằng lệnh:

Kết nối tới Qdrant database ở Docker

```
1 client = QdrantClient(url="http://localhost:6333")
```

6. Tiếp theo, chúng ta sẽ tạo một collection tên `mcp_database` để lưu trữ tất cả dữ liệu vào collection này. Một collection trong Qdrant là một tập hợp các points (vectors với payload) mà bạn có thể tìm kiếm. Mỗi vector của các point trong cùng một collection phải có cùng kích thước và được so sánh bằng một metric duy nhất (ví dụ: cosine similarity, Euclidean distance, v.v.). Các vector được đặt tên có thể được sử dụng để có nhiều vector trong một point, mỗi vector có thể có yêu cầu về kích thước và metric riêng. Chúng ta tạo collection này bằng cách sử dụng method `create_collection` của `client` như sau:

Tạo collection mcp_database trong Qdrant client

```
1 client.create_collection(
2     collection_name="mcp_database",
3     vectors_config=models.VectorParams(
4         size=embedding_model.get_sentence_embedding_dimension(),
5         distance=models.Distance.COSINE
6     )
7 )
```

Trong đó, các arguments được giải thích như sau:

- `collection_name="mcp_database"`: Tên của collection được tạo là `mcp_database`.
- `vectors_config`: Định nghĩa cấu hình cho các vector sẽ lưu trữ trong collection này.
 - `size=embedding_model.get_sentence_embedding_dimension()`: Độ dài của mỗi vector embedding, lấy từ mô hình embedding đang sử dụng.
 - `distance=models.Distance.COSINE`: Phương thức tính toán độ tương đồng giữa hai embedding vectors. Chúng ta sử dụng cosine similarity trong bài demo này.

7. Sau khi kết nối tới Qdrant Database bằng một trong hai cách trên và tạo collection `mcp_database`, chúng ta lưu các embeddings cùng với các documents tương ứng vào database với method `upload_points`:

Lưu dữ liệu embedding của documents cùng với nội dung tương ứng vào Qdrant database

```

1 client.upload_points(
2     collection_name="mcp_database",
3     points=[
4         models.PointStruct(
5             id=idx, vector=document_embeddings[idx],
6             payload={
7                 "text": doc["text"],
8                 "path": doc["path"],
9                 "source": doc.get("source", ""),
10                "split": doc.get("split", ""),
11                "id": doc.get("id", "")
12            }
13        ) for idx, doc in enumerate(documents)
14    ],
15)

```

Trong đó, mỗi point đại diện cho một document với các attribute như sau:

- `id=idx`: ID duy nhất của point, được gán bằng chỉ số của document trong danh sách.
- `vector=document_embeddings[idx]`: Vector embedding tương ứng với document.
- `payload`: Chứa các thông tin metadata của document như:
 - `text`: Nội dung văn bản của document.
 - `path`: Đường dẫn tới file lưu trữ document trên ổ cứng.
 - `source`: Nguồn gốc của document (nếu có).
 - `split`: Phân chia dữ liệu (nếu có).
 - `id`: ID của sample trong dataset (nếu có).

III.4. Xây dựng Retriever

Chúng ta tạo file mới `retriever.py` để định nghĩa các hàm và class cần thiết cho việc truy vấn dữ liệu từ Qdrant Database. Trong file này, chúng ta sẽ định nghĩa class `Retriever` với các hàm thêm văn bản vào database và truy vấn văn bản từ database. Class này sẽ sử dụng Qdrant Client để tương tác với Qdrant Database và thực hiện các thao tác tìm kiếm.

- Đầu tiên, chúng ta thêm các thư viện cần thiết vào file `retriever.py`:

Import các thư viện cần thiết của retriever

```

1 import logging
2 import uuid
3 from typing import Any, Dict, List, Optional
4
5 from qdrant_client import QdrantClient
6 from qdrant_client.http.models import Distance, PointStruct, VectorParams
7 from sentence_transformers import SentenceTransformer

```

- Tiếp theo, chúng ta định nghĩa class `Retriever` với các methods để thêm văn bản vào database và truy vấn văn bản từ database:

Class Retriever với các methods tương tác với cơ sở dữ liệu

```

1 class Retriever:
2     def __init__(
3         self,
4         embedding_model: SentenceTransformer,
5         qdrant_path: str,
6         collection_name: str = "documents",
7         embedding_size: Optional[int] = None,
8     ):
9         """
10            Initializes the Retriever with the embedding model, Qdrant path,
11            collection name, and embedding size.
12
13        Args:
14            embedding_model (SentenceTransformer): The embedding model to use for
15                encoding documents.
16            qdrant_path (str): The path to the Qdrant database (local or URL).
17            collection_name (str): The name of the collection in Qdrant to store
18                documents.
19            embedding_size (Optional[int]): The size of the embedding vectors. If
20                None, it will be inferred from the model.
21
22        Raises:
23            ValueError: If embedding_size is not a positive integer.
24
25        self.embedding_model = embedding_model
26        self.qdrant_path = qdrant_path
27        self.collection_name = collection_name

```

```

24     # Connect to Qdrant local instance (using the local path)
25     self.client = QdrantClient(
26         path=self.qdrant_path,
27     )
28
29     # If embedding_size is not provided, attempt to infer it
30     if embedding_size is None:
31         embedding_size = self.embedding_model.get_sentence_embedding_dimension()
32     else:
33         if embedding_size <= 0:
34             raise ValueError("Vector size must be a positive integer.")
35     self.embedding_size = embedding_size
36
37     # Ensure the collection exists
38     if not self.client.collection_exists(self.collection_name):
39         self.client.create_collection(
40             collection_name=self.collection_name,
41             vectors_config=VectorParams(size=embedding_size, distance=Distance.COSINE),
42         )
43
44     def add_documents(self, documents: list[str]) -> Dict[str, str]:
45         ...
46
47     def add_document(self, document: str) -> Dict[str, str]:
48         ...
49
50     def retrieve(self, query: str, limit: int = 5) -> List[dict]:
51         ...

```

Trong đó, các methods được implement như sau:

- `__init__`: Khởi tạo class Retriever với các arguments như mô hình embedding, đường dẫn tới Qdrant Database, tên collection và kích thước embedding. Nếu kích thước embedding không được cung cấp, nó sẽ được lấy từ mô hình embedding.
- `add_documents`: Thêm một danh sách các documents vào Qdrant Database. Mỗi document sẽ được chuyển đổi thành embedding và lưu trữ trong collection với một UUID duy nhất.

Method `add_documents` để thêm nhiều documents vào database

```

1 ...
2
3 def add_documents(self, documents: list[str]) -> Dict[str, str]:
4     """
5         Embeds the documents and adds them to the Qdrant collection.
6         Each document is assigned a unique UUID.
7     """
8     try:
9         embeddings = self.embedding_model.encode(documents)

```

```

10
11     points = [
12         PointStruct(
13             id=str(uuid.uuid4()), # unique id for each document
14             vector=embedding,
15             payload={"text": doc},
16         )
17         for doc, embedding in zip(documents, embeddings)
18     ]
19     self.client.upsert(collection_name=self.collection_name, points=
20                         points)
21     return {"status": "success", "message": "Documents added successfully
22                         ."}
23 except Exception as e:
24     logging.error(f"Error adding documents: {e}")
25     return {"status": "error", "message": str(e)}
26 ...

```

- **add_document:** Thêm một document đơn lẻ vào Qdrant Database. Tương tự như phương thức trên nhưng chỉ xử lý một document.

Method add_document để thêm một document riêng lẻ vào database

```

1 ...
2
3 def add_document(self, document: str) -> Dict[str, str]:
4     """
5     Embeds a single document and adds it to the Qdrant collection.
6     The document is assigned a unique UUID.
7     """
8     try:
9         embedding = self.embedding_model.encode([document])[0]
10        point = PointStruct(
11            id=str(uuid.uuid4()), # unique id for the document
12            vector=embedding,
13            payload={"text": document},
14        )
15        self.client.upsert(collection_name=self.collection_name, points=[point])
16        return {"status": "success", "message": "Document added successfully.
17                         ."}
18    except Exception as e:
19        logging.error(f"Error adding document: {e}")
20        return {"status": "error", "message": str(e)}
21 ...

```

- **retrieve:** Truy vấn các documents tương tự với query đầu vào với metric cosine similarity. Method này trả về một danh sách các dictionary chứa nội dung của document và độ tương đồng giữa query với document tương ứng.

Method retrieve để tìm documents có nội dung tương ứng trong database

```

1 ...
2
3 def retrieve(self, query: str, limit: int = 5) -> List[dict]:
4     """
5         Retrieves documents similar to the query using cosine similarity.
6         Returns a list of dictionaries containing the document text and its score
7
8     """
9     query_embedding = self.embedding_model.encode([query])[0].tolist()
10    search_result = self.client.query_points(
11        collection_name=self.collection_name,
12        query=query_embedding,
13        limit=limit,
14    )
15
16    results = [
17        {"text": point.payload["text"], "score": point.score} for point in
18            search_result.points
19    ]
20    return results
21
22 ...

```

3. Để kiểm tra logic của retriever, chúng ta test các hàm này trong file retriever.py với hàm như sau:

Kiểm tra Retriever đã build bằng hàm main trong Python

```

1 if __name__ == "__main__":
2     embedding_model = SentenceTransformer(
3         "Alibaba-NLP/gte-multilingual-base", trust_remote_code=True, cache_folder=
4             "./cache"
5     )
6     retriever = Retriever(
7         embedding_model=embedding_model,
8         qdrant_path="./qdrant_database",
9         collection_name="mcp_database",
10    )
11
12    results = retriever.retrieve("What is organic chemistry?", limit=5)
13    for i, result in enumerate(results):
14        print(f"Result {i + 1} (Score: {result['score']:.4f}): {result['text'][:250]}...")

```

Cuối cùng, chúng ta chạy file retriever.py với Python bằng lệnh sau trong terminal:

```
python retriever.py
```

và nhận được output là các kết quả truy vấn từ Qdrant Database với các documents tương tự với query “What is organic chemistry?”. Các kết quả này sẽ được sắp xếp theo điểm số tương ứng, phục vụ cho việc thiết kế các logic tìm kiếm nâng cao ở phía MCP server.

Output khi chạy đoạn code trên

Result 1 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 2 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 3 (Score: 0.7766): Bioorganic chemistry is a scientific discipline that combines organic chemistry and biochemistry. It is that branch of life science that deals with the study of biological processes using chemical methods. Protein and enzyme function are examples of ...

Result 4 (Score: 0.7752): Organic Chemistry

Organic chemistry is a branch of chemistry that deals with the study of the structure, properties, and reactions of organic compounds and materials. These compounds contain carbon atoms and are found in various forms, including liv...

Result 5 (Score: 0.7617): Organic chemistry is a subdiscipline within chemistry involving the scientific study of the structure, properties, and reactions of organic compounds and organic materials, i.e., matter in its various forms that contain carbon atoms. Study of structu...

III.5. Xây dựng MCP Server

Cuối cùng, chúng ta tạo file `server.py` để định nghĩa MCP Server của chúng ta. Các chức năng của Personal Database MCP Server sẽ bao gồm:

- **Tools:** Cung cấp các tools để thực hiện các tác vụ sau:
 - Truy vấn các documents từ Qdrant Database dựa trên query của người dùng,
 - Tìm kiếm query của người dùng trên internet bằng công cụ DuckDuckGo Search (nếu cơ sở dữ liệu cá nhân không có thông tin cần thiết),
 - Lưu trữ một document vào cơ sở dữ liệu cá nhân (Qdrant Database) và lưu xuống local file system để sử dụng sau này,
- **Resources:** Cung cấp các resources sau đây để lấy documents từ cơ sở dữ liệu cá nhân:
 - Lấy danh sách các topics trong cơ sở dữ liệu cá nhân,
 - Lấy tất cả các documents trong một topic cụ thể,
 - Lấy danh sách các documents trong một topic cụ thể với phân trang,

- **Prompts:** Cung cấp các mẫu prompt templates để hỗ trợ việc sử dụng MCP server này hiệu quả hơn, bao gồm các prompts có chức năng như sau:
 - Prompt để truy vấn các documents từ Qdrant Database,
 - Prompt để tìm kiếm thông tin từ internet,
 - Prompt để thêm một document vào cơ sở dữ liệu cá nhân,

Chúng ta sẽ sử dụng thư viện FastMCP được cung cấp trong framework mcp để xây dựng Personal Library MCP Server theo các bước dưới đây:

1. Đầu tiên, chúng ta import các packages cần thiết cho việc xây dựng MCP server:

Import thư viện để xây dựng Personal Library MCP Server

```

1 import logging
2 import os
3 from typing import Any, Dict, List, Optional
4
5 import click
6 from langchain_community.tools import DuckDuckGoSearchResults
7 from mcp.server.fastmcp import FastMCP
8 from mcp.server.fastmcp.prompts import base
9 from pydantic import BaseModel, Field
10 from sentence_transformers import SentenceTransformer
11 from retriever import Retriever
12 from create_vector_database import (
    discover_and_get_all_files,
    load_json_file,
    load_txt_file,
    load_md_file,
)
13 import uuid
14 import json

```

2. Chúng ta thiết kế hàm run_mcp_server để khởi tạo cơ sở dữ liệu, load các documents và chạy MCP Server như sau:

Hàm run_mcp_server để khởi tạo và chạy Personal Database MCP Server

```

1 def run_mcp_server():
2     embedding_model = SentenceTransformer(
3         "Alibaba-NLP/gte-multilingual-base",
4         trust_remote_code=True,
5         cache_folder=".cache",
6     )
7     qdrant_path = "./qdrant_database"
8
9     retriever = Retriever(
10         embedding_model=embedding_model,
11         qdrant_path=qdrant_path,
12         collection_name="mcp_database",

```

```

13 )
14
15 DOCUMENT_DIR = "./documents"
16 DOCUMENTS_PATH_BY_TOPICS = []
17 for folder in os.listdir(DOCUMENT_DIR):
18     if os.path.isdir(os.path.join(DOCUMENT_DIR, folder)):
19         DOCUMENTS_PATH_BY_TOPICS[folder] = discover_and_get_all_files(
20             os.path.join(DOCUMENT_DIR, folder),
21             allowed_extensions=[".json", ".txt", ".md"],
22             recursive=True,
23         )
24
25 ALL_DOCUMENT_PATHS = []
26 for folder in DOCUMENTS_PATH_BY_TOPICS:
27     ALL_DOCUMENT_PATHS.extend(DOCUMENTS_PATH_BY_TOPICS[folder])
28
29 mcp_server = create_mcp_server(retriever, DOCUMENTS_PATH_BY_TOPICS,
30                                 ALL_DOCUMENT_PATHS)
31 mcp_server.run(transport="streamable-http")

```

Hàm `run_mcp_server` sẽ thực hiện các bước sau:

- Khởi tạo mô hình embedding `embedding_model` sử dụng thư viện `sentence_transformers` với model `Alibaba-NLP/gte-multilingual-base`.
- Khởi tạo Qdrant Database với đường dẫn `qdrant_database` và tên collection `mcp_database`.
- Load các document paths từ thư mục `documents` vào hai dictionary:
 - `DOCUMENTS_PATH_BY_TOPICS`: Dictionary chứa danh sách các document paths theo tên topic,
 - `ALL_DOCUMENT_PATHS`: Danh sách tất cả các document paths.
- Khởi tạo MCP server với các tools, resources và prompts bằng hàm `create_mcp_server`.
- Chạy MCP server với transport `streamable-http`.

- Tiếp theo, chúng ta định nghĩa các tools, resources và prompts cho MCP server trong hàm `create_mcp_server`:

Hàm `create_mcp_server` để khởi tạo Personal Database MCP Server

```

1 def create_mcp_server(
2     retriever: Retriever,
3     documents_path_by_topics: Dict[str, List[str]],
4     all_document_paths: List[str],
5 ):
6     mcp = FastMCP(
7         name="Retriever MCP Server",
8         host="127.0.0.1",

```

```

9     port=2545,
10    description="A server that provides MCP-powered agentic RAG capabilities."
11
12    ,
13
14    ...
15
16    return mcp

```

Chúng ta sẽ đi thiết kế lần lượt các tools, resources và prompts cho MCP Server để điền vào dấu ... trong hàm `create_mcp_server`.

4. Tools

Để xây dựng tools cho MCP Server với FastMCP, ta sử dụng `@tool` decorator với các arguments như sau:

- `name`: Tên tùy chọn cho công cụ (mặc định sẽ dùng tên của function)
- `title`: Tiêu đề tùy chọn, dễ hiểu dành cho con người
- `description`: Mô tả tùy chọn về chức năng của công cụ
- `annotations`: ToolAnnotations tùy chọn, cung cấp thông tin bổ sung cho công cụ
- `structured_output`: Điều khiển việc đầu ra của công cụ có được cấu trúc hay không
 - Nếu là `None`: tự động phát hiện dựa trên kiểu dữ liệu trả về được chú thích trong function
 - Nếu là `True`: luôn tạo tool với đầu ra có cấu trúc (nếu kiểu trả về cho phép)
 - Nếu là `False`: luôn tạo tool với đầu ra không có cấu trúc

(a) Tool tìm kiếm documents từ Qdrant Database

Chúng ta sẽ sử dụng tool này để truy vấn các documents từ Qdrant Database dựa trên query của người dùng. Input của tool này tuân theo schema sau:

Schema QueryInput

```

1 class QueryInput(BaseModel):
2     query: str = Field(description="The query to retrieve relevant documents.
3                                         ")
4     num_documents: int = Field(default=5, description="The number of
5                                         documents to retrieve for the query.")

```

Tool `retrieve_documents_from_database` được định nghĩa như sau:

Tool retrieve_documents_from_database

```

1 @mcp.tool(
2     name="retrieve_documents_from_database",

```

```

3     title="Retrieve Documents from Database",
4     description="Retrieve documents similar to the query from the database.",
5 )
6 def retrieve(input: QueryInput) -> RetrievalResult:
7     retrieval_results = retriever.retrieve(input.query, input.num_documents)
8     return RetrievalResult(
9         results=[
10            RetrievedDocument(text=result["text"], score=result["score"])
11            for result in retrieval_results
12        ]
13    )

```

Khi tool được gọi, method `retrieve` trong class `Retriever` sẽ được chạy để truy vấn các documents từ Qdrant Database. Tool này sẽ trả về kết quả theo schema `RetrievalResult`, chứa danh sách các documents tuân theo schema `RetrievedDocument` gồm nội dung của văn bản với điểm số tương ứng:

Output schema của tool `retrieve_documents_from_database`

```

1 class RetrievedDocument(BaseModel):
2     text: str = Field(description="The content of the retrieved document.")
3     score: Optional[float] = Field(None, description="The score of the
4                                     retrieved document based on similarity
5                                     .")
6
7 class RetrievalResult(BaseModel):
8     results: List[RetrievedDocument] = Field(description="List of retrieved
9                                              documents with their scores.")

```

(b) Tool tìm kiếm query trên internet

Để thiết kế tool `search_query_on_internet` này, ta dùng class `DuckDuckGoSearchResults` từ thư viện `langchain_community.tools`.

Đầu vào của tool này là query của người dùng và số lượng documents cần tìm kiếm giống như tool `retrieve_documents_from_database` ở trên. Tiếp theo, chúng ta implement tool này như sau:

Tool `search_query_on_internet`

```

1 @mcpc.tool(
2     name="search_query_on_internet",
3     title="Search the Query on the Internet",
4     description="Search for documents on the internet related to the query.
5                 This tool will be used when the
6                 database does not have relevant
7                 documents for the query.",
8 )
9 def search_query_on_internet(input: QueryInput) -> SearchResult:
10    search_engine = DuckDuckGoSearchResults(

```

```
8     output_format="list", num_results=input.num_documents, backend="text"
9 )
10 search_results = search_engine.invoke(input.query)
11 return SearchResult(
12     results=[
13         RetrievedDocument(
14             text=f"Title: {result['title']}\nText: {result['snippet']}",
15             score=None,
16         )
17         for result in search_results
18     ]
19 )
```

Khi được thực thi, tool sẽ gọi method `invoke` trong class `DuckDuckGoSearchResults` để tìm kiếm query trên internet với các arguments `output_format="list"` (kết quả trả về là một list of documents) và `num_results` là số lượng documents cần tìm kiếm.

Tương tự như tool `retrieve_documents_from_database`, tool này sẽ trả về một danh sách các documents `SearchResult` có liên quan đến query đầu vào dưới dạng danh sách các documents `RetrievedDocument` với điểm số tương ứng.

(c) Tool lưu trữ một document vào cơ sở dữ liệu cá nhân

Chúng ta thiết kế tool `add_document_to_database` để lưu trữ một document vào cơ sở dữ liệu cá nhân:

Tool add_document_to_database

```
1 @mcp.tool(
2     name="add_document_to_database",
3     title="Add a Document to Database",
4     description="Add a document to the database for future retrieval.",
5 )
6 def add_document_to_database(
7     document: str, topic_name: Optional[str] = None, document_name: Optional[
8         str] = None
9 ) -> AddDocumentResponse:
10     doc_id = str(uuid.uuid4())
11     if topic_name is None:
12         topic_name = "default"
13     if not os.path.exists(f"./documents/{topic_name}"):
14         os.makedirs(f"./documents/{topic_name}")
15     if document_name is None:
16         document_name = doc_id
17
18     # save to the documents folder
19     with open(f"./documents/{topic_name}/{document_name}.json", "w") as f:
20         doc_data = {
21             "text": document,
22             "path": f"./documents/{topic_name}/{document_name}.json",
23             "source": "user-added",
24             "split": topic_name,
```

```

24         "id": doc_id,
25     }
26     json.dump(doc_data, f, indent=4)
27
28     # add to the database
29     retriever.add_document(doc_data)
30
31     return AddDocumentResponse(status="success", message="Document added to
database.")

```

Hàm được xây dựng như sau:

- Tool nhận vào ba arguments:
 - **document**: Nội dung của document cần lưu trữ,
 - **topic_name**: Tên topic của document,
 - **document_name**: Tên của document.
- Tiếp theo, chúng ta tiến hành xử lí bổ sung các thông tin của document như sau:
 - Tạo một UUID duy nhất cho document,
 - Nếu **topic_name** không được cung cấp, chúng ta sẽ sử dụng giá trị mặc định là "**default**",
 - Nếu **document_name** không được cung cấp, chúng ta sẽ sử dụng UUID làm tên của document,
 - Tạo thư mục cho topic nếu chưa tồn tại,
- Tiếp theo, chúng ta lưu document vào thư mục topic với tên là **document_name** dưới dạng file **.json**:
 - Tạo một dictionary chứa các thông tin của document,
 - Lưu dictionary này vào file **{document_name}.json** trong thư mục **{topic_name}**,
- Cuối cùng, chúng ta thêm document vào Qdrant Database với method **add_document** trong class **Retriever** và trả về kết quả.

5. Resources

Trong Personal Database MCP Server, resources của chúng ta là những documents được lưu trong folder **documents** với các định dạng **.json**, **.txt** và **.md** (chủ yếu là **.json**). Do đó, chúng ta sẽ thiết kế các MCP resources để lấy các resources này.

Để xây dựng resources cho MCP Server với FastMCP, ta sử dụng **@resource** decorator với các arguments như sau:

- **uri**: URI của resource (có thể chứa các arguments như **{topic_name}**, **{page_number}**), ví dụ như **document://topics** hoặc **document://topics/{topic_name}**
- **name**: Tên tùy chọn cho resource (mặc định sẽ dùng tên của function định nghĩa

resource)

- **title:** Tiêu đề tùy chọn, dễ hiểu dành cho người dùng
- **description:** Mô tả tùy chọn về chức năng của resource
- **mime_type:** MIME type tùy chọn cho resource, ví dụ như `text/plain`, `application/json`, `application/pdf`, v.v.

(a) Resources lấy danh sách các topics trong cơ sở dữ liệu cá nhân

Chúng ta thiết kế resource `get_all_topics` với URI `document://topics` để lấy danh sách các topics trong cơ sở dữ liệu cá nhân:

Resource document://topics

```

1 @mcp.resource(
2     uri="document://topics",
3     name="get_all_topics",
4     title="Get All Topics in the Database",
5     description="A resource to get all topics in the database.",
6 )
7 def get_all_topics() -> List[str]:
8     return sorted(list(documents_path_by_topics.keys()))

```

Khi được gọi, resource này sẽ trả về danh sách các topics trong cơ sở dữ liệu cá nhân.

(b) Resources lấy tất cả các documents trong một topic cụ thể

Để lấy tất cả các documents trong một topic cụ thể, chúng ta sẽ sử dụng resource `get_all_documents_by_topic` với URI `document://topics/{topic_name}`:

Resource document://topics/{topic_name}

```

1 @mcp.resource(
2     uri="document://topics/{topic_name}",
3     name="get_all_documents_by_topic",
4     title="Get All Documents by Topic",
5     description="A resource to get all documents by topic.",
6 )
7 def get_all_documents_by_topic(topic_name: str) -> LocalDocumentList:
8     doc_paths = documents_path_by_topics[topic_name]
9     documents = []
10    for doc_path in doc_paths:
11        if doc_path.endswith(".json"):
12            doc = load_json_file(doc_path)
13            doc = LocalDocument(
14                text=doc["text"],
15                path=doc_path,
16                source=doc.get("source", ""),
17                split=doc.get("split", ""),
18                id=doc.get("id", ""),
19            )
20        elif doc_path.endswith(".txt"):

```

```

21         doc = LocalDocument(
22             text=load_txt_file(doc_path),
23             path=doc_path,
24             source=None,
25             split=None,
26             id=None,
27         )
28     elif doc_path.endswith(".md"):
29         doc = LocalDocument(
30             text=load_md_file(doc_path),
31             path=doc_path,
32             source=None,
33             split=None,
34             id=None,
35         )
36     else:
37         continue
38     documents.append(doc)
39 return LocalDocumentList(documents=documents)

```

Resource `get_all_documents_by_topic` được lấy bằng URI `document://topics/{topic_name}` với argument `topic_name` là tên của topic cần lấy documents. Khi được gọi, resource này sẽ lấy toàn bộ đường dẫn các documents trong topic cụ thể, đọc nội dung của từng document và trả về danh sách các documents `LocalDocument` với các thông tin của document, bao gồm `text`, `path`, `source`, `split` và `id`:

Output schema của resource `get_all_documents_by_topic`

```

1 class LocalDocument(BaseModel):
2     text: str = Field(description="The content of the local document.")
3     path: str = Field(description="The path of the local document.")
4     source: Optional[str] = Field(None, description="The source of the local
5                                     document.")
6     split: Optional[str] = Field(None, description="The split of the local
7                                   document.")
8     id: Optional[str] = Field(None, description="The id of the local document
9                               .")
10
11 class LocalDocumentList(BaseModel):
12     documents: List[LocalDocument] = Field(description="The list of local
13                                              documents.")

```

(c) Resources lấy các documents trong một topic cụ thể với cơ chế phân trang

Cơ chế phân trang được sử dụng để tránh việc lấy quá nhiều documents cùng lúc, đặc biệt là khi số lượng documents trong một topic quá nhiều.

Để lấy các documents của một topic với phân trang, chúng ta sẽ sử dụng resource `get_documents_by_topic` với URI `document://topics/{topic_name}/pages/{page_number}`:

Resource document://topics/{topic_name}/pages/{page_number}

```

1 @mcp.resource(
2     uri="document://topics/{topic_name}/pages/{page_number}",
3     name="get_documents_by_topic",
4     title="Get Documents by Topic",
5     description="A resource to get documents by topic using pagination. Each
6                 page contains 10 documents.",
7 )
8 def get_documents_by_topic(topic_name: str, page_number: int) ->
9                 LocalDocumentList:
10            doc_paths = documents_path_by_topics[topic_name]
11            start_index, end_index = (page_number - 1) * 10, page_number * 10
12            documents = []
13            for doc_path in doc_paths[start_index:end_index]:
14                if doc_path.endswith(".json"):
15                    doc = load_json_file(doc_path)
16                elif doc_path.endswith(".txt"):
17                    doc = LocalDocument(
18                        text=load_txt_file(doc_path),
19                        path=doc_path,
20                        source=None,
21                        split=None,
22                        id=None,
23                    )
24                elif doc_path.endswith(".md"):
25                    doc = LocalDocument(
26                        text=load_md_file(doc_path),
27                        path=doc_path,
28                        source=None,
29                        split=None,
30                        id=None,
31                    )
32                else:
33                    continue
34            documents.append(doc)
35 return LocalDocumentList(documents=documents)

```

Resource `get_documents_by_topic` được lấy bằng URI `document://topics/{topic_name}/pages/{page_number}` với argument `topic_name` là tên của topic cần lấy documents và `page_number` là vị trí trang cần lấy. Khi được gọi, resource này sẽ trả về tối đa 10 documents. Kết quả trả về tuân theo kiểu dữ liệu `LocalDocumentList`, chia danh sách các documents `LocalDocument` với các thông tin của document, bao gồm `text`, `path`, `source`, `split` và `id` tương tự như kết quả trả về của resource `get_all_documents_by_topic`.

6. Prompts

Bên cạnh tools và resources, chúng ta cũng thiết kế thêm các prompts để hỗ trợ người dùng sử dụng Personal Database MCP Server.

Chúng ta sẽ viết các prompt cho MCP server của chúng ta bằng `@prompt()` decorator với các arguments sau:

- **name:** Tên tùy chọn cho prompt (mặc định là tên của hàm)
- **title:** Tiêu đề hiển thị dễ hiểu dành cho con người (tùy chọn)
- **description:** Mô tả tùy chọn về chức năng hoặc mục đích của prompt. Nếu field này bị bỏ trống, phần docstring của hàm định nghĩa prompt sẽ được sử dụng thay thế.

(a) **Prompt retrieve_documents_from_database_prompt**

Prompt này được thiết kế để lấy các documents từ cơ sở dữ liệu cá nhân như sau:

```
Prompt retrieve_documents_from_database_prompt

1 @mcp.prompt(
2     name="retrieve_documents_from_database_prompt",
3     title="Retrieve Documents from Database Prompt",
4     description="Prompt to retrieve documents similar to the query from the
5                 database.",
6 )
7 def get_retrieve_document_from_database_tool(input: QueryInput) -> base.
8                 UserMessage:
9                 return base.UserMessage(
10                    content=f """Answer the following question. Remember to use the 'retrive_documents_from_database' tool
11                     to find {input.num_documents}
12                     relevant documents in the database to support your answer.
13
14 Question: {input.query}
15 """
16             )
```

Prompt nhận vào argument `input` là kiểu dữ liệu `QueryInput`, gồm query và số lượng documents cần lấy, và trả về một message `UserMessage` chứa nội dung prompt.

(b) **Prompt retrieve_document_and_search_internet_prompt** để lấy các documents từ cơ sở dữ liệu cá nhân và tìm kiếm trên internet nếu cần:

```
Prompt retrieve_document_and_search_internet_prompt

1 @mcp.prompt(
2     name="retrieve_document_and_search_internet_prompt",
3     title="Retrieve Document and Search Internet Prompt",
4     description="Prompt to retrieve documents similar to the query from the
5                 database and search the internet if necessary.",
6 )
7 def get_retrieve_document_and_search_internet_tool(input: QueryInput) -> base.
8                 UserMessage:
9                 return base.UserMessage(
10                    content=f """Answer the following question. Remember to use the 'retrive_documents_from_database' tool
11                     to find {input.num_documents}"""
12             )
```

```

9
10 Question: {input.query}
11 """
12 )

```

relevant documents in the database to support your answer. If the database does not have relevant documents, use the 'search_documents_on_internet' tool to search for documents on the internet.

Tương tự như prompt `retrieve_documents_from_database_prompt`, prompt này cũng nhận vào argument `input` là kiểu dữ liệu `QueryInput`, gồm query và số lượng documents cần lấy, và trả về một message `UserMessage` chứa nội dung prompt. Tuy nhiên, prompt này còn thêm câu hỏi nếu cơ sở dữ liệu không có documents liên quan, thì sẽ sử dụng tool `search_documents_on_internet` để tìm kiếm trên internet.

- (c) Prompt `search_query_on_internet_prompt` để tìm kiếm trên internet trực tiếp (không cần lấy từ cơ sở dữ liệu cá nhân):

Prompt `search_query_on_internet_prompt`

```

1 @mcp.prompt(
2     name="search_query_on_internet_prompt",
3     title="Search Internet Prompt",
4     description="Prompt to search the internet for documents related to the
5         query.",
6 )
7 def get_search_internet_tool(input: QueryInput) -> base.UserMessage:
8     return base.UserMessage(
9         content=f"""
10            Answer the following question. Remember to use the 'search_documents_on_internet' tool to
11            find {input.num_documents} relevant
12            documents on the internet to support
13            your answer.
14
15 Question: {input.query}
16 """
17     )

```

Giống với prompt `retrieve_documents_from_database_prompt`, prompt này cũng nhận vào argument `input` là kiểu dữ liệu `QueryInput`, gồm query và số lượng documents cần lấy, và trả về một message `UserMessage` chứa nội dung prompt. Tuy nhiên, prompt này không sử dụng tool `retrieve_documents_from_database` để lấy documents từ cơ sở dữ liệu cá nhân, mà sẽ tìm kiếm trên internet trực tiếp.

- (d) Prompt `add_single_document_to_database_prompt` để thêm một document vào cơ sở dữ liệu cá nhân:

Prompt add_single_document_to_database_prompt

```

1 @mcp.prompt(
2     name="add_single_document_to_database_prompt",
3     title="Add Single Document to Database Prompt",
4     description="Prompt to add a single document to the database.",
5 )
6 def get_add_single_document_to_database_tool(document: str) -> base.
7     UserMessage:
8         return base.UserMessage(
9             content=f """Add the following document to the database for future
10            retrieval using the 'add_single_document_to_database' tool.
11
12             Document: {document}
13 """
14         )

```

Prompt nhận vào argument `document` là nội dung của document cần thêm vào cơ sở dữ liệu cá nhân, và trả về một message `UserMessage` chứa nội dung prompt.

III.6. Chạy MCP Server và tích hợp vào Claude Desktop

Run MCP Server Để chạy MCP Server, chúng ta có thể thực thi một trong các lệnh sau trong terminal:

- Chạy bằng uv: `uv run server.py`
- Chạy bằng python: `python server.py`

Kết quả chạy MCP server sẽ hiển thị như sau:

Output trong terminal khi chạy MCP server

```

INFO:      Started server process [175875]
INFO:      Waiting for application startup.
2025-07-13 11:18:23,876 - mcp.server.streamable_http_manager - INFO -
    StreamableHTTP session manager started
INFO:      Application startup complete.
INFO:      Uvicorn running on http://127.0.0.1:2545 (Press CTRL+C to quit)

```

Chúng ta sử dụng URL `http://127.0.0.1:2545` để kết nối với MCP Server ở phần tiếp theo.

Test MCP Server Để test MCP Server, chúng ta sử dụng tool MCP Inspector. Cách chạy tool này đã được trình bày chi tiết ở mục [II.6.5..](#) Chúng ta bật trình duyệt và truy cập vào URL có dạng: `http://localhost:6274/?MCP_PROXY_AUTH_TOKEN=...` trong output của terminal để sử dụng MCP Inspector.

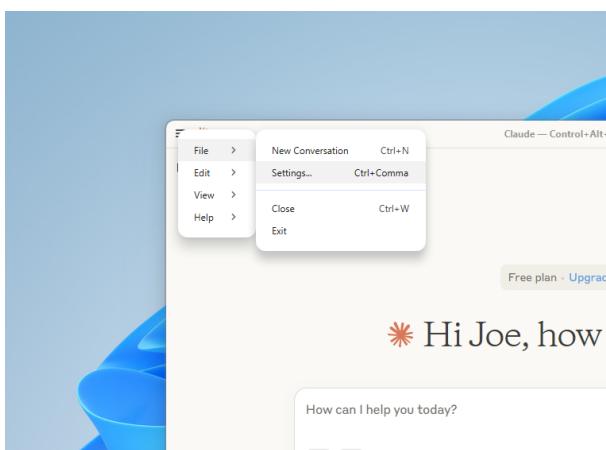
Ở thanh sidebar bên trái, chúng ta nhập vào các thông số như sau để kết nối với MCP Server:

- Transport Type: Chọn Streamable HTTP,
- URL: <http://127.0.0.1:2545>.

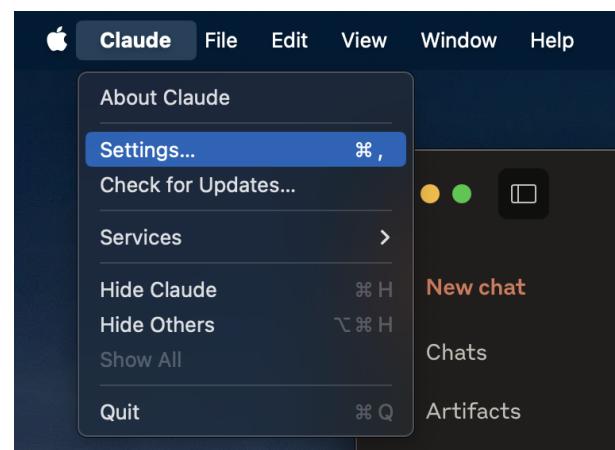
và ấn nút Connect để kết nối với MCP Server. Sau khi kết nối thành công, chúng ta có thể sử dụng các tool của MCP Server để test như trong hướng dẫn ở mục [II.6.5..](#)

Tích hợp MCP Server vào Claude Desktop Để thêm MCP Server vào Claude Desktop, chúng ta thực hiện theo các bước sau:

1. Mở Claude Desktop và mở Setting (xem Hình [18](#)),



(a) Claude Desktop Setting on Windows

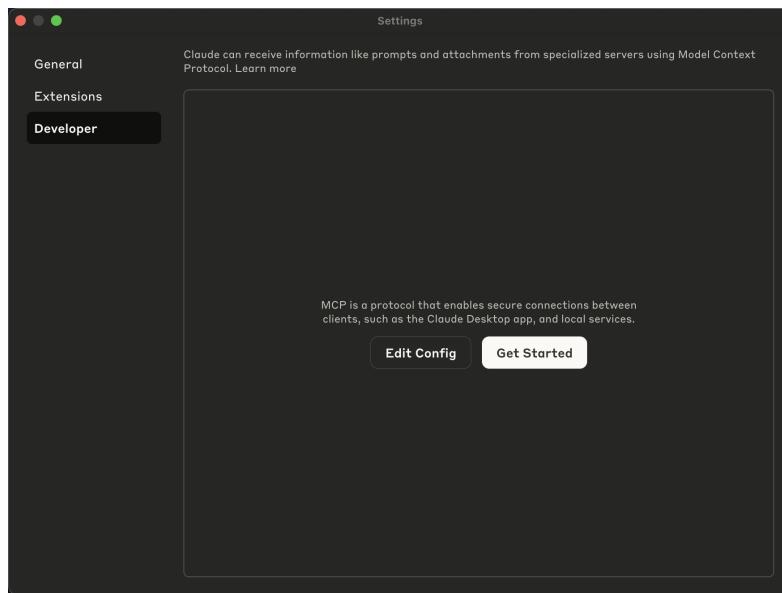


(b) Claude Desktop Setting on MacOS

Hình 18: Claude Desktop Setting

2. Trong tab Developer, chọn button Edit Config (xem Hình [19](#)) để tạo file `claude_desktop_config.json` trong thư mục sau:

- Window: `%APPDATA%\Claude\claude_desktop_config.json`
- MacOS: `/Library/Application/Support/Claude/claude_desktop_config.json`



Hình 19: Developer Tab in Claude Config

3. Chúng ta thêm đoạn code sau vào file `claude_desktop_config.json`:

Nội dung file config claude_desktop_config.json khi MCP server dùng Streamable HTTP transport

```
{
  "mcpServers": {
    "Personal Database MCP Server": {
      "command": "npx",
      "args": ["mcp-remote"],
      "→ "http://127.0.0.1:2545/mcp"
    }
  }
}
```

Trong đó, `http://127.0.0.1:2545/mcp` là URL đến MCP server của chúng ta. Nếu MCP server được deploy trên một máy chủ cloud, chúng ta có thể thay `127.0.0.1` bằng public IP của máy chủ cloud đó.

Ngoài ra, nếu chúng ta sử dụng stdio transport (định nghĩa ở phần code 3, dòng 33 với giá trị của argument `transport="stdio"`), chúng ta sử dụng đoạn code sau trong file config của Claude Desktop (thay `/ABSOLUTE/PATH/TO/OUR/PROJECT/DIRECTORY` thành đường dẫn đến project folder của chúng ta):

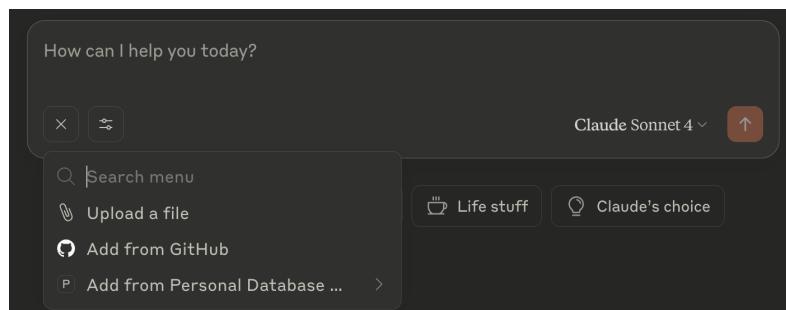
Nội dung file config claude_desktop_config.json khi MCP server dùng stdio transport

```
{
  "mcpServers": [
    "Personal Database MCP Server": {
      "command": "uv",
      "args": [
        "--directory",
        "/ABSOLUTE/PATH/TO/OUR/PROJECT/DIRECTORY",
        "run",
        "server.py"
      ]
    }
  ]
}
```

- Khởi động lại Claude Desktop

III.7. Sử dụng Prompts, Resources và Tools trên Claude Desktop

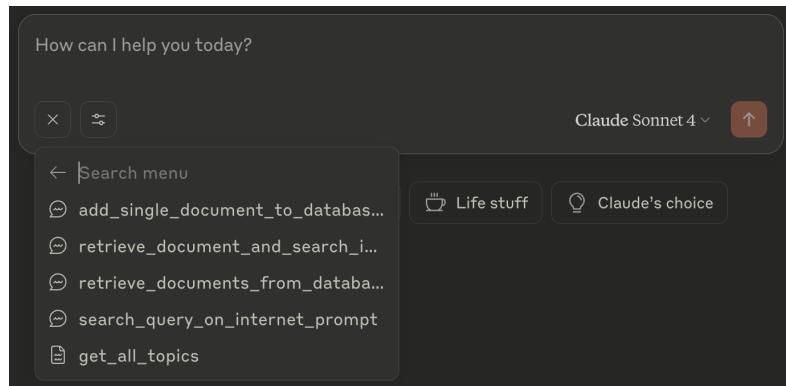
Resources Chúng ta xem danh sách các prompts của server bằng cách ấn vào nút + trong góc trái dưới của hộp thoại. Danh sách Resources và Prompts của từng MCP server sẽ hiển thị dưới dạng dropdown menu và phía dưới option Add from Github (xem Hình 20).



Hình 20: Danh sách các MCP Server

Sau đó, chúng ta chọn Personal Database MCP Server để xem danh sách các prompts và resources (xem Hình 21), bao gồm:

- Prompts: `retrieve_document_from_database`, `retrieve_document_and_search_internet`, `search_query_on_internet`, `add_single_document_to_database`
- Resources: `get_all_topics`



Hình 21: Danh sách Prompts và Resource của Personal Database MCP Server trong Claude Desktop

Chúng ta chọn resource `get_all_topics` để lấy danh sách các topics trong cơ sở dữ liệu cá nhân. Kết quả trả về sẽ được đính kèm trong hộp thoại chat của Claude Desktop như Hình 22a. Khi ấn vào file, nội dung sẽ được hiển thị như Hình 22b.

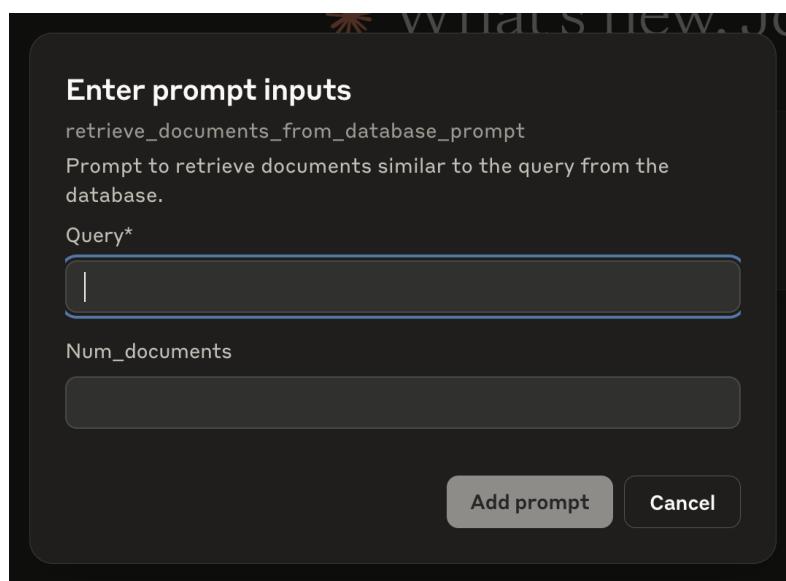
```
[ "art_and_culture_textbook",
  "art_and_culture_wiki",
  "biology_textbook",
  "biology_wiki",
  "chemistry_textbook",
  "chemistry_wiki",
  "default",
  "history_textbook",
  "history_wiki",
  "medicine_textbook",
  "medicine_wiki",
  "pedagogy_textbook",
  "pedagogy_wiki",
  "philosophy_textbook",
  "philosophy_wiki",
  "physics_textbook",
  "physics_wiki",
  "psychiatry_textbook",
  "psychiatry_wiki",
  "psychology_textbook",
  "psychology_wiki",
  "religion_textbook",
  "religion_wiki",
  "science_studies_textbook",
  "science_studies_wiki",
  "social_studies_textbook",
  "social_studies_wiki" ]
```

(a) Kết quả trả về khi lấy resource được đính kèm trong hộp thoại

(b) Xem nội dung của resource được trả về

Hình 22: Kết quả resource trả về từ MCP server được đính vào trong hộp thoại chat với Claude

Prompt Chúng ta chọn prompt `retrieve_document_from_database` và một popup hiển thị để chúng ta nhập vào các arguments của prompt đó (xem Hình 23).

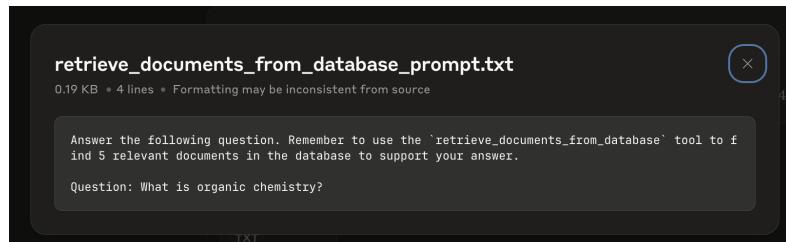


Hình 23: Popup window để nhập arguments của prompt

Chúng ta nhập các field như sau:

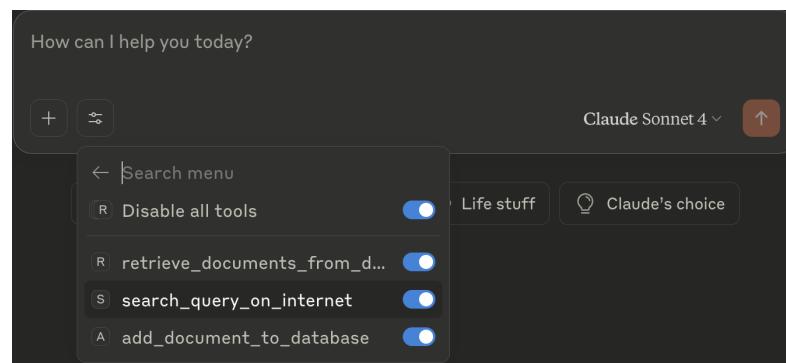
- Query: What is organic chemistry?
- Num_documents: 5

Sau khi nhập xong, ta ấn Add prompt. Prompt trả về được đính kèm vào ô hộp thoại tương tự như khi thêm resource. Nội dung của prompt được trả về được hiển thị trong Hình 24.



Hình 24: Nội dung của prompt khi được đọc từ ô hộp thoại

Tools Sau khi có prompt `retrieve_document_from_database` được trả về trong hộp thoại, chúng ta gửi đoạn chat cho Claude. Lưu ý rằng chúng ta phải bật tool `retrieve_document_from_database` trong danh sách tools của Personal Database MCP Server (xem Hình 25).



Hình 25: Danh sách các tools của Personal Database MCP Server

Lúc này, Claude sẽ xử lí message của chúng ta như sau:

- Đầu tiên, mô hình AI sẽ ra tín hiệu gọi tool `retrieve_document_from_database` trong quá trình xử lí message của chúng ta.
- Claude Desktop (AI application) xin quyền được gọi tool từ người dùng (xem Hình 26a). Trong popup xin quyền gọi tool, các arguments để gọi tool cũng được đính kèm.
- Sau khi được cấp quyền sử dụng tool, MCP server sẽ chạy tool và trả về các văn bản liên quan được đính kèm trong cuộc hội thoại (xem Hình 26b) để mô hình AI sinh câu trả lời cho người dùng.

(a) Claude Desktop xin quyền để gọi tool

(b) Kết quả gọi tool

Hình 26: Dùng tools trong Claude Desktop

Toàn bộ conversation (bao gồm arguments gọi tool và tool response) có thể được xem ở [đây](#).

IV. Câu hỏi trắc nghiệm

IV.1. Câu hỏi

1. Trong kiến trúc của MCP, thành phần nào chịu trách nhiệm điều phối và kiểm soát logic nghiệp vụ cấp cao?
 - (a) MCP Server.
 - (b) MCP Client.
 - (c) MCP Host.
 - (d) Transport Layer.
2. MCP client duy trì kết nối với MCP server theo mô hình nào?
 - (a) Nhiều-một.
 - (b) Một-nhiều.
 - (c) Một-một.
 - (d) Nhiều-nhiều.
3. Định dạng nào được MCP sử dụng để truyền thông điệp giữa client và server?
 - (a) gRPC.
 - (b) HTTP/2.
 - (c) JSON-RPC 2.0.
 - (d) SOAP.
4. Trong số các cơ chế truyền tải của MCP, cơ chế nào phù hợp cho môi trường local?
 - (a) WebSocket Transport.
 - (b) Streamable HTTP Transport.
 - (c) Server-Sent Events.
 - (d) Stdio Transport.
5. Vai trò chính của MCP server là gì?
 - (a) Tổng hợp dữ liệu từ các MCP Client.
 - (b) Hiển thị giao diện người dùng.
 - (c) Cung cấp năng lực chuyên biệt thông qua giao thức chuẩn.
 - (d) Lưu trữ toàn bộ phiên làm việc của Host.
6. MCP server cung cấp các loại tài nguyên nào sau đây? (Chọn phương án đúng nhất)
 - (a) Resources, Plugins, Queries.
 - (b) Documents, Tools, Commands.

- (c) Files, Streams, Tasks.
(d) Resources, Tools, Prompts.
7. Tool trong MCP Server khác với Resource ở điểm nào?
(a) Tools chỉ dành cho quản trị viên.
(b) Tools có thể thực thi và thay đổi trạng thái server.
(c) Resources được gọi tự động bởi LLM.
(d) Resources có thể bị sửa đổi bởi client.
8. Trong quá trình handshake giữa MCP Client và Server, điều gì xảy ra?
(a) Truy vấn resources từ server.
(b) Gửi kết quả trả lời của LLM.
(c) Thương lượng các tính năng được hỗ trợ giữa hai bên.
(d) Tạo tệp log hệ thống.
9. Prompt trong MCP Server được dùng để:
(a) Gửi thông điệp bảo mật giữa client và server.
(b) Làm mẫu hướng dẫn mô hình AI hoạt động theo ý định định sẵn.
(c) Tạo file output từ AI application.
(d) Giao tiếp giữa các MCP Servers.
10. Trong kiến trúc MCP, khi nào Host sẽ khởi tạo MCP Client mới?
(a) Khi kết nối với mạng Internet.
(b) Khi cần cập nhật mô hình AI.
(c) Khi bắt đầu kết nối đến một MCP server.
(d) Khi hệ thống gặp lỗi kết nối.

IV.2. Đáp án và giải thích

1. Đáp án đúng: (C) MCP Host

Giải thích: MCP Host là thành phần chủ động khởi tạo kết nối tới MCP Server, kiểm soát quyền truy cập, quản lý client và điều phối toàn bộ logic nghiệp vụ cấp cao của ứng dụng AI.

2. Đáp án đúng: (C) Một-một

Giải thích: Mỗi MCP client duy trì một kết nối 1:1 với một MCP server cụ thể để đảm bảo an toàn và không chồng chéo dữ liệu giữa các server khác nhau.

3. Đáp án đúng: (C) JSON-RPC 2.0

Giải thích: MCP sử dụng định dạng JSON-RPC 2.0 để truyền và xử lý thông điệp giữa client và server vì tính đơn giản, tiêu chuẩn và khả năng hỗ trợ các yêu cầu có trạng thái.

4. Đáp án đúng: (D) Stdio Transport

Giải thích: Stdio Transport sử dụng luồng chuẩn đầu vào/đầu ra để giao tiếp và không yêu cầu mạng, rất phù hợp với môi trường cục bộ như terminal.

5. Đáp án đúng: (C) Cung cấp năng lực chuyên biệt thông qua giao thức chuẩn

Giải thích: MCP Server hoạt động như một adapter, đóng gói và công bố các năng lực chuyên biệt (như truy vấn, xử lý dữ liệu) thông qua một giao diện chuẩn.

6. Đáp án đúng: (D) Resources, Tools, Prompts

Giải thích: Đây là ba thành phần cốt lõi (primitives) của MCP Server: Resource cung cấp dữ liệu, Tool là hành động có thể thực thi, Prompt là chỉ dẫn cho mô hình AI.

7. Đáp án đúng: (B) Tools có thể thực thi và thay đổi trạng thái server

Giải thích: Khác với resource (chỉ đọc), tool trong MCP server có khả năng thay đổi trạng thái hệ thống thông qua hành động như thêm, xoá hoặc cập nhật dữ liệu.

8. Đáp án đúng: (C) Thương lượng các tính năng được hỗ trợ giữa hai bên

Giải thích: Quá trình handshake giữa MCP client và MCP server giúp thống nhất về các tính năng như mã hoá, định dạng message, tool có sẵn,... trước khi giao tiếp chính thức bắt đầu.

9. Đáp án đúng: (B) Làm mẫu hướng dẫn mô hình AI hoạt động theo ý định định sẵn

Giải thích: Prompt là các template hoặc hướng dẫn định nghĩa sẵn nhằm định hình cách mô hình AI phản hồi, thường được sử dụng trong những tương tác có cấu trúc lặp lại.

10. Đáp án đúng: (C) Khi cần truy cập một MCP Server mới với chức năng riêng biệt

Giải thích: MCP Host sẽ tạo MCP client mới để kết nối đến mỗi MCP server riêng biệt, đảm bảo mỗi dịch vụ (như lưu trữ tệp, truy vấn dữ liệu) được tách biệt và an toàn.

V. Tài liệu tham khảo

Introduction, Definitions, and Examples

- [1] A. PBC, *Introducing the Model Context Protocol*, <https://www.anthropic.com/news/model-context-protocol>.
- [2] M. C. Protocol, *Introduction*, <https://modelcontextprotocol.io/introduction>.
- [3] D. N. Koul, *The Model Context Protocol (MCP) - A Complete Tutorial*, <https://medium.com/@nimritakoul01/the-model-context-protocol-mcp-a-complete-tutorial-a3abe8a7f4ef>.
- [4] A. P. Avi Chawla, *The Full MCP Blueprint: Background, Foundations, Architecture, and Practical Usage (Part A)*, <https://www.dailydoseofds.com/model-context-protocol-crash-course-part-1/>.

Function Calling

- [5] OpenAI, *Function Calling*, <https://platform.openai.com/docs/guides/function-calling>.
- [6] Google, *Function calling with the Gemini API*, <https://ai.google.dev/gemini-api/docs/function-calling>.
- [7] Z. Huang, *MCP Simply Explained: Function Calling Rebranded or Genuine Breakthrough?* <https://dev.to/zachary62/model-context-protocol-mcp-simply-explained-function-calling-rebranded-or-genuine-breakthrough-4c04>.
- [8] A. Chawla, *The M*N Integration Problem Solved by MCP*, <https://blog.dailydoseofds.com/p/the-mn-integration-problem-solved>.

Architecture and Components

- [10] M. C. Protocol, *Core Architecture*, <https://modelcontextprotocol.io/docs/concepts/architecture>.
- [11] M. C. Protocol, *Architecture*, <https://modelcontextprotocol.io/specification/2025-06-18/architecture>.
- [12] M. C. Protocol, *Sampling*, <https://modelcontextprotocol.io/docs/concepts/sampling>, <https://modelcontextprotocol.io/specification/2025-06-18/client/sampling>.
- [13] M. C. Protocol, *Elicitation*, <https://modelcontextprotocol.io/docs/concepts/elicitation>, <https://modelcontextprotocol.io/specification/2025-06-18/client/sampling>.
- [14] M. C. Protocol, *Roots*, <https://modelcontextprotocol.io/docs/concepts/roots>, <https://modelcontextprotocol.io/specification/2025-06-18/client/elicitation>.
- [15] M. C. Protocol, *Overview*, <https://modelcontextprotocol.io/specification/2025-06-18/server>.

- [16] M. C. Protocol, *Resources*, <https://modelcontextprotocol.io/docs/concepts/resources>.
- [17] M. C. Protocol, *Tools*, <https://modelcontextprotocol.io/docs/concepts/tools>.
- [18] M. C. Protocol, *Prompts*, <https://modelcontextprotocol.io/docs/concepts/prompts>.
- [19] modelcontextprotocol, *MCP Simple Streamable Http Stateless Server Example*, <https://github.com/modelcontextprotocol/python-sdk/tree/main/examples/servers/simple-streamablehttp-stateless>.
- [20] modelcontextprotocol, *MCP Python SDK/Advanced Usage/Low-Level Server*, <https://github.com/modelcontextprotocol/python-sdk>.
- [21] M. Paktiti, *How MCP servers work: Components, logic, and architecture*, <https://workos.com/blog/how-mcp-servers-work>.
- [22] FastMCP, *MCP Transport*, <https://gofastmcp.com/clients/transports>.

Visualization

- [9] A. Chawla, *The MCP Illustrated Guidebook*, <https://blog.dailydoseofds.com/p/the-mcp-illustrated-guidebook>.
- [23] S. Wang, *What is the Use of Claude's MCP (Model Context Protocol)?* <https://wshuyi.medium.com/what-is-the-use-of-claudes-mcp-model-context-protocol-3ac863fb4c70>.
- [24] C. Greyling, *Model Context Protocol (MCP)*, <https://cobusgreyling.medium.com/model-context-protocol-mcp-da3e0f912bbc>.
- [25] A. K. G. Rao, *Building Custom Tools With Model Context Protocol*, <https://dzone.com/articles/building-custom-tools-model-context-protocol>.
- [26] A. Chawla, *Visual Guide to Model Context Protocol (MCP)*, <https://blog.dailydoseofds.com/p/visual-guide-to-model-context-protocol>.
- [27] Flaticon, *Flaticon Attributes: The figures includes resources from Flaticon.com*, <https://www.flaticon.com/>.

Implementation

- [28] A. PBC, *Model Context Protocol Server Collections*, <https://github.com/modelcontextprotocol/servers>.
- [29] FastMCP, *Connect UV Python MCP Server to Claude Desktop*, <https://gofastmcp.com/integrations/clause-desktop>.
- [30] D. O'Dell, *Build a fully remote MCP multi-tool server with FastMCP SSE transport and Cursor integration*, <https://medium.com/@texasdave2/build-a-fully-remote-mcp-tool-server-with-fastmcp-sse-transport-and-cursor-integration-ec4ab0a3f01e>.
- [31] M. AI, *Tìm hiểu, xây dựng MCP Server và kết nối với Claude.ai, Agent*, <https://www.youtube.com/watch?v=6rDUTy87Xxg>.

- [32] theailanguage, *MCP Course 11 - Build your first Streamable HTTP FastMCP server, run with Claude Desktop*, <https://www.youtube.com/watch?v=j63xRV52IMk>.
- [33] M. C. Protocol, *MCP Inspector*, <https://github.com/modelcontextprotocol/inspector>.
- [34] M. Inspector, *An LLM-Powered Chatbot MCP Client written in Python*, <https://github.com/modelcontextprotocol/quickstart-resources/tree/main/mcp-client-python>.
- [35] M. C. Protocol, *MCP Server Simple Tool*, <https://github.com/modelcontextprotocol/python-sdk/tree/b16c2a85189075fe060ba0136ce4a34ec776cbb7/examples/servers/simple-tool>.

Phụ lục

1. Toàn bộ source code trong bài có thể được tải tại [đây](#).