MCPClientLLM

October 24, 2025

[1]:
```
#!pip install -q -U google-genai
#!pip install fastmcp
```

0.1 Setup Gemini API client

[41]:
```
import os
from google import genai
from dotenv import load_dotenv, find_dotenv

# Read the local .env file, containing the Gemini secret API key.
_ = load_dotenv(find_dotenv())

gemini_client = genai.Client(api_key = os.environ["GEMINI_API_KEY"])
```

0.2 Setup and test MCP client

[42]:
```
import asyncio
import fastmcp
import json

# Connect to HTTP MCP server.
mcp_client = fastmcp.Client("http://localhost:8000/mcp")

async def main():
    async with mcp_client:
        # Basic server interaction.
        await mcp_client.ping()

        # List available tools.
        tools = await mcp_client.list_tools()
        for tool in tools:
            print(tool)
        resources = await mcp_client.list_resources()
        print(resources)
        prompts = await mcp_client.list_prompts()
        print(prompts)
```

```
        # Execute tools.
        result1 = await mcp_client.
 ↪call_tool("get_restaurants_for_location_cuisine",
                                              {"city": "Charlotte", "state":␣
 ↪"NC", "cuisine": "Korean"})
        candidates = json.loads(result1.content[0].text)

        print(f"Found {len(candidates)} candidate restaurants.")
        for rest in candidates:
            print(rest)
            result2 = await mcp_client.call_tool("is_restaurant_kid_friendly",
                                                 {"name": rest["name"],
                                                  "full_address":␣
 ↪(rest["street_address"], rest["city"], rest["state"])})
            print(f"{rest["name"]} is " + ("" if result2.content else "not ") +␣
 ↪"kid friendly.")


# asyncio.run(main())
await(main())
```

```
name='get_restaurants_for_location_cuisine' title=None description='Gets a list
of restaurants that are located at a particular location (city and state) and
serve a particular cuisine.\n    ' inputSchema={'properties': {'city': {'type':
'string'}, 'state': {'type': 'string'}, 'cuisine': {'type': 'string'}},
'required': ['city', 'state', 'cuisine'], 'type': 'object'}
outputSchema={'properties': {'result': {'items': {}, 'type': 'array'}},
'required': ['result'], 'type': 'object', 'x-fastmcp-wrap-result': True}
icons=None annotations=None meta={'_fastmcp': {'tags': []}}
name='is_restaurant_kid_friendly' title=None description='Returns True if a
restaurant is kid friendly, False otherwise.\n    ' inputSchema={'properties':
{'name': {'title': 'Name'}, 'full_address': {'title': 'Full Address'}},
'required': ['name', 'full_address'], 'type': 'object'} outputSchema=None
icons=None annotations=None meta={'_fastmcp': {'tags': []}}
[]
[]
Found 1 candidate restaurants.
{'name': 'Soy Bistro', 'city': 'Charlotte', 'state': 'NC', 'street_address':
'5008 Maryland Way', 'postal_code': '37027'}
Soy Bistro is kid friendly.
```

0.3 MCP Tool Use with Gemini

[33]:
```
from google import genai
from google.genai import types

# Define the function declaration for the model
```

```
get_restaurant_tool = {
    "name": "get_restaurants_for_location_cuisine",
    "description": "Gets a list of restaurants that are located at a particular␣
 ↪location (city and state) and serve a particular cuisine.",
    "parameters": {
        "type": "object",
        "properties": {
            "city": {
                "type": "string",
                "description": "The city name, e.g. San Francisco",
            },
            "state": {
                "type": "string",
                "description": "The state (two letter abbreviation), e.g. CA",
            },
            "cuisine": {
                "type": "string",
                "description": " The type of cuisine, e.g. Seafood",
            },
        },
        "required": ["city", "state", "cuisine"],
    },
}

tools = types.Tool(function_declarations = [get_restaurant_tool])
config = types.GenerateContentConfig(tools = [tools])

# Make the request. The SDK handles the function call and returns the final␣
 ↪response.
response = gemini_client.models.generate_content(
    model = "gemini-2.5-flash",
    contents = "Can you show me all restaurants in Philadelphia, PA, that offer␣
 ↪Thai food?",
    config = config
)

# Check for a function call,
if response.candidates[0].content.parts[0].function_call:
    function_call = response.candidates[0].content.parts[0].function_call
    print(f"Function to call: {function_call.name}")
    print(f"Arguments: {function_call.args}")
    async with mcp_client:
        result = await mcp_client.call_tool(function_call.name, function_call.
 ↪args)
        candidates = json.loads(result.content[0].text)
        print(f"Found {len(candidates)} candidate restaurants.")
        for rest in candidates:
```

```
            print(rest)
else:
    print("No function call found in the response.")
    print(response.text)
```

```
Function to call: get_restaurants_for_location_cuisine
Arguments: {'state': 'PA', 'city': 'Philadelphia', 'cuisine': 'Thai'}
Found 2 candidate restaurants.
{'name': 'New South Ocean Chinese & Japanese Restaurant', 'city':
'Philadelphia', 'state': 'PA', 'street_address': '1664 Bethlehem Pike',
'postal_code': '19031'}
{'name': 'Amber Asian Cafe', 'city': 'Philadelphia', 'state': 'PA',
'street_address': '860 W Main St', 'postal_code': '19446'}
```

[40]:
```
# Define the function declaration for the model
is_kid_friendly_tool = {
    "name": "is_restaurant_kid_friendly",
    "description": "Returns True if a restaurant is kid friendly, False␣
 ↪otherwise.",
    "parameters": {
        "type": "object",
        "properties": {
            "name": {
                "type": "string",
                "description": "The restaurant name",
            },
            "street_address": {
                "type": "string",
                "description": "The street address, e.g. 9201 University Blvd",
            },
            "city": {
                "type": "string",
                "description": "The city name, e.g. San Francisco",
            },
            "state": {
                "type": "string",
                "description": "The state (two letter abbreviation), e.g. CA",
            },
        },
        "required": ["name", "street_address", "city", "state"],
    },
}

tools = types.Tool(function_declarations = [is_kid_friendly_tool])
config = types.GenerateContentConfig(tools = [tools])
```

```
# Make the request. The SDK handles the function call and returns the final␣
 ↪response.
response = gemini_client.models.generate_content(
    model = "gemini-2.5-flash",
    contents = "Can you tell me if I can bring my kids to the restaurant Amber␣
 ↪Asian Cafe, which is located at 860 W Main St in Philadelphia, PA?",
    config = config
)

# Check for a function call,
if response.candidates[0].content.parts[0].function_call:
    function_call = response.candidates[0].content.parts[0].function_call
    print(f"Function to call: {function_call.name}")
    print(f"Arguments: {function_call.args}")

    full_address = (function_call.args["street_address"], function_call.
 ↪args["city"], function_call.args["state"])
    name = function_call.args["name"]

    async with mcp_client:
        result = await mcp_client.call_tool(function_call.name,
                                            {"name": rest["name"],␣
 ↪"full_address": full_address})
        print("The restaurant is " + ("" if result.content else "not ") + "kid␣
 ↪friendly.")
else:
    print("No function call found in the response.")
    print(response.text)
```

```
Function to call: is_restaurant_kid_friendly
Arguments: {'street_address': '860 W Main St', 'state': 'PA', 'city':
'Philadelphia', 'name': 'Amber Asian Cafe'}
The restaurant is kid friendly.
```

0.4 Use weather MCP server to get forecast at `longitude` and `latitude`

Even better:

1. First use another MCP server to find the longitude and latitude for a certain address `city` and `state`.
2. Then use the weather MCP server to get the weather.

[ ]:
```
# YOUR CODE HERE
```