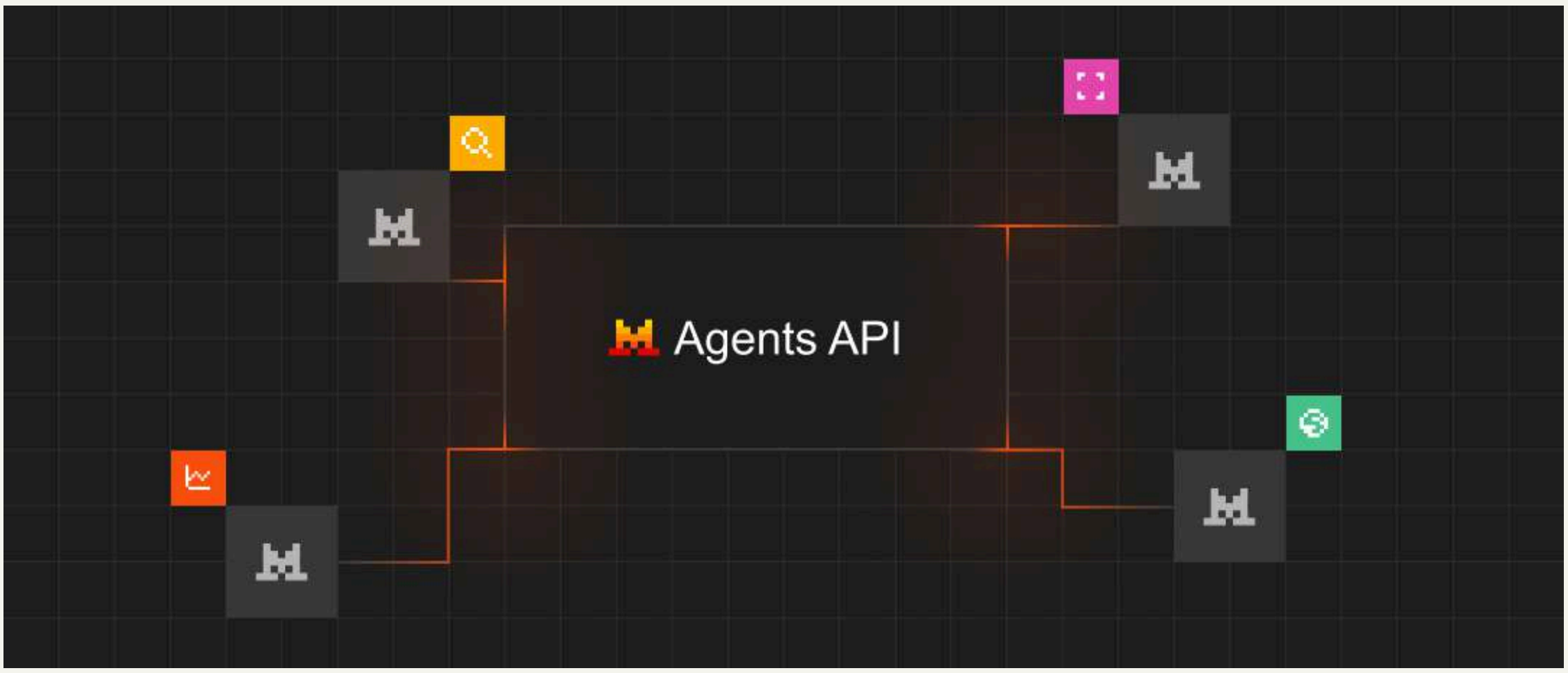
Guide to building Agents with Mistral Agents API

Dipanjan (DJ)

What is Mistral Agents API



- **AI agents are autonomous systems powered by LLMs that, given high-level instructions, can plan, use tools, carry out processing steps, and take actions to achieve specific goals.**

- **Mistral Agents API allows developers to build such agents, leveraging multiple features such as:**
  - Multiple mutlimodal models available, text and vision models.
  - Persistent state across conversations.
  - Ability to have conversations with base models, a single agent, and multiple agents.
  - Built-in connector tools for code execution, web search, image generation and document library out of the box.
  - Handoff capability to use different agents as part of a workflow, allowing agents to call other agents.
  - Features supported via chat completions endpoint are also supported, such as:
    - Structured Outputs
    - Document Understanding
    - Tool Usage
    - Citations

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agent Creation Basics

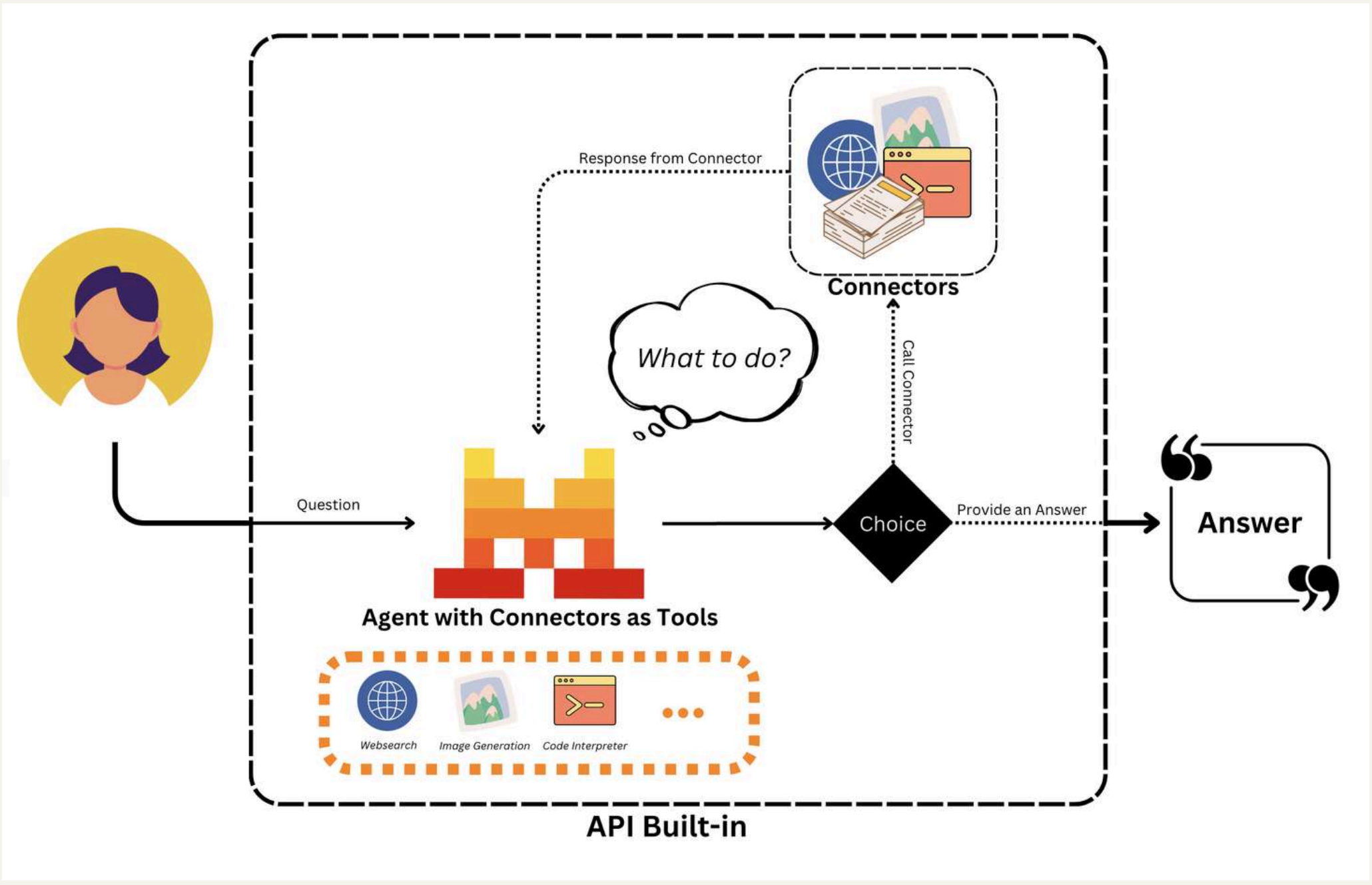
Here is an example of a Web Search Agent using our built-in tool:

```
websearch_agent = client.agents.create(
    model="mistral-medium-2505",
    description="Agent able to search information over the web, such as news, weather, sport results...",
    name="Websearch Agent",
    instructions="You have the ability to perform web searches with `web_search` to find up-to-date information.",
    tools=[{"type": "web_search"}],
    completion_args={
        "temperature": 0.3,
        "top_p": 0.95,
    }
)
```

- **When creating an Agent, there are multiple parameters and values that need to be set in advance. These are:**
  - **model:** The model your agent will use among our available models for chat completion.
  - **description:** The agent description, related to the task it must accomplish or the use case at stake.
  - **name:** The name of your agent.
  - **instructions:** The main instructions of the agent, also known as the system prompt. This must accurately describe the main task of your agent. (optional)
  - **tools:** A list of tools the model can make use of. There are currently different types of tools (optional):
    - **function:** User-defined tools, with similar usage to the standard function calling used with chat completion.
    - **web_search/web_search_premium:** Built-in tool for web search.
    - **code_interpreter:** Built-in tool for code execution.
    - **image_generation:** Built-in tool for image generation.
- **completion_args:** Standard chat completion sampler arguments. All chat completion arguments are accepted (optional).

Source: Mistral Agents Docs

Agent Connectors or Tools



- **Connectors are tools that Agents can call at any given point. They are deployed and ready for the agents to leverage to answer questions on demand.**
- **They are also available for users to use them directly via Conversations without the Agent creation step!**
- **You can create an Agent with the tools as follows:**

```
library_agent = client.beta.agents.create(
    model="...",
    name="...",
    description="...",
    instructions="...",
    tools=[...]
)
```

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents with Custom Tools

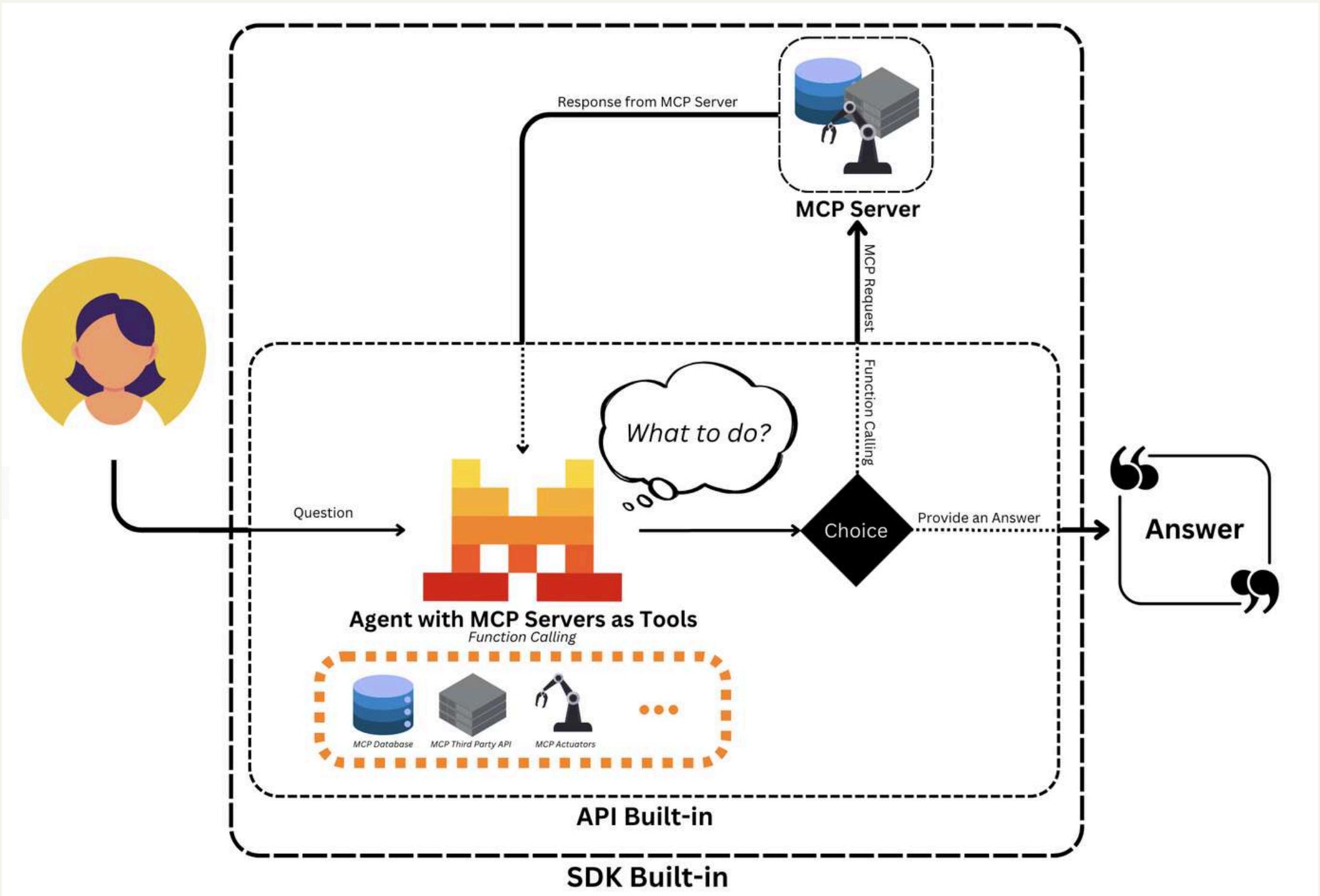
```python
from typing import Dict

# Define custom tool
def get_european_central_bank_interest_rate(date: str) -> Dict[str, str]:
    """
    Retrieve the interest rate of the European Central Bank for a given date.

    Parameters:
    - date (str): The date for which to retrieve the interest rate in the format YYYY-MM-DD.

    Returns:
    - dict: A dictionary containing the date and the corresponding interest rate.
    """
    # This is a mock implementation.
    # In a real scenario, you would fetch this data from an API or database.
    interest_rate = "2.5%"
    return {
        "date": date,
        "interest_rate": interest_rate
    }

# Create the Agent and add custom tool
ecb_interest_rate_agent = client.beta.agents.create(
    model="mistral-medium-2505",
    description="Can find the current interest rate of the European central bank",
    name="ecb-interest-rate-agent",
    tools=[
        {
            "type": "function",
            "function": {
                "name": "get_european_central_bank_interest_rate",
                "description": "Retrieve the interest rate of European central bank.",
                "parameters": {
                    "type": "object",
                    "properties": {
                        "date": {"type": "string",},
                    },
                    "required": ["date",]
                },
            },
        },
    ],
)
```

- **Define custom tools as python functions**
- **Add them with schema details in the agent tools section**

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents with MCP



- **The Model Context Protocol (MCP) is an open standard designed to streamline the integration of AI models with various data sources and tools**

- **By providing a standardized interface, MCP enables seamless and secure connections, allowing AI systems to access and utilize contextual information efficiently**

- **Mistral has a nice Python SDK which enables seamless integration of their agents with MCP Clients.**

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents with MCP

Step 1: Initialize the Mistral Client

First, we import everything needed. Most of the required modules are available with our `mistralai` package, but you will also need `mcp`. All the MCP Clients will be run asynchronously, so we will create an async main function where the main code will reside.

```python
#!/usr/bin/env python
import asyncio
import os

from mistralai import Mistral
from mistralai.extra.run.context import RunContext
from mcp import StdioServerParameters
from mistralai.extra.mcp.stdio import MCPClientSTDIO
from pathlib import Path

from mistralai.types import BaseModel

# Set the current working directory and model to use
cwd = Path(__file__).parent
MODEL = "mistral-medium-latest"

async def main() -> None:
    # Initialize the Mistral client with your API key
    api_key = os.environ["MISTRAL_API_KEY"]
    client = Mistral(api_key)
```

Step 2: Define Server Parameters and Create an Agent

We can now define the server parameters, which will point to a specific path. For more information, we recommend visiting the Model Context Protocol documentation. Once the server is defined, we can create our agent.

```python
    # Define parameters for the local MCP server
    server_params = StdioServerParameters(
        command="python",
        args=[str((cwd / "mcp_servers/stdio_server.py").resolve())],
        env=None,
    )

    # Create an agent to tell the weather
    weather_agent = client.beta.agents.create(
        model=MODEL,
        name="weather teller",
        instructions="You are able to tell the weather.",
        description="",
    )
```

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents with MCP

Step 3: Define Output Format and Create a Run Context

The next step is to create a Run Context where everything will happen between the MCP Client and our Agent. You can also leverage structured outputs!

```
    # Define the expected output format for weather results
    class WeatherResult(BaseModel):
        user: str
        location: str
        temperature: float

    # Create a run context for the agent
    async with RunContext(
        agent_id=weather_agent.id,
        output_format=WeatherResult,
        continue_on_fn_error=True,
    ) as run_ctx:
```

Step 4: Register MCP Client

The next step is to create and register the MCP Client.

```
        # Create and register an MCP client with the run context
        mcp_client = MCPClientSTDIO(stdio_params=server_params)
        await run_ctx.register_mcp_client(mcp_client=mcp_client)
```

You can also leverage the MCP Orchestration to use Function Calling locally directly.

```
    import random
    # Register a function to get a random location for a user, it will be an available tool
    @run_ctx.register_func
    def get_location(name: str) -> str:
        """Function to get location of a user.

        Args:
            name: name of the user.
        """
        return random.choice(["New York", "London", "Paris", "Tokyo", "Sydney"])

    # Create and register an MCP client with the run context
    mcp_client = MCPClientSTDIO(stdio_params=server_params)
    await run_ctx.register_mcp_client(mcp_client=mcp_client)
```

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents with MCP

Step 5: Run the Agent and Print Results

Everything is ready; you can run our Agent and get the output results!

```
        # Run the agent with a query
        run_result = await client.beta.conversations.run_async(
            run_ctx=run_ctx,
            inputs="Tell me the weather in John's location currently.",
        )

        # Print the results
        print("All run entries:")
        for entry in run_result.output_entries:
            print(f"{entry}")
            print()
        print(f"Final model: {run_result.output_as_model}")

if __name__ == "__main__":
    asyncio.run(main())
```

Streaming Conversations

Streaming conversations with an agent using a local MCP server is similar to non-streaming, but instead of waiting for the entire response, you process the results as they arrive.

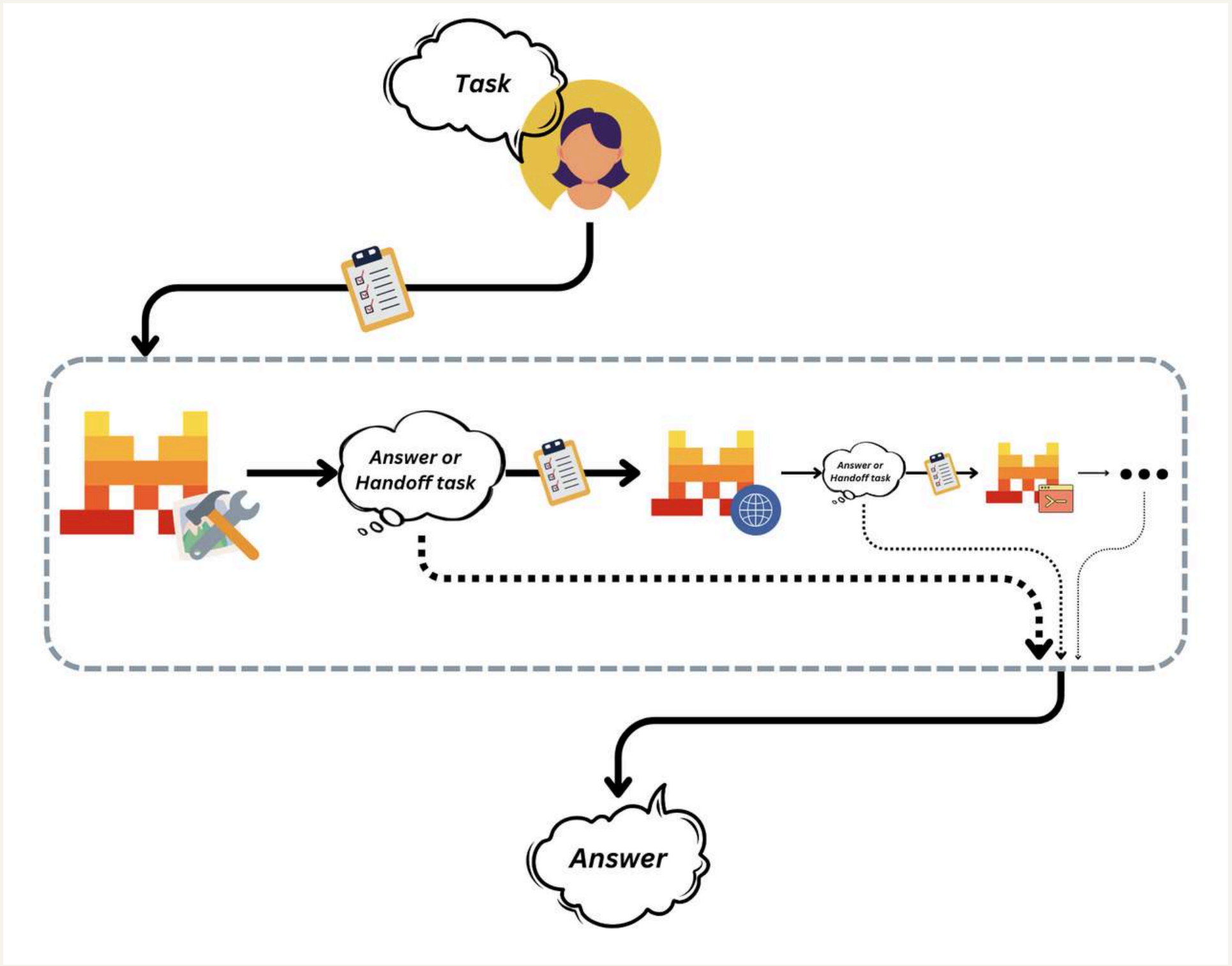
Here is a brief example of how to stream conversations:

```
    # Stream the agent's responses
    events = await client.beta.conversations.run_stream_async(
        run_ctx=run_ctx,
        inputs="Tell me the weather in John's location currently.",
    )

    # Process the streamed events
    run_result = None
    async for event in events:
        if isinstance(event, RunResult):
            run_result = event
        else:
            print(event)

    if not run_result:
        raise RuntimeError("No run result found")

    # Print the results
    print("All run entries:")
    for entry in run_result.output_entries:
        print(f"{entry}")
    print(f"Final model: {run_result.output_as_model}")
```

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents Handoffs



- **When creating and using Agents, often with access to specific tools, there are moments where it is desired to call other Agents mid-action.**

- **To elaborate and engineer workflows for diverse tasks that you may want automated, this ability to give Agents tasks or hand over a conversation to other agents is called Handoffs.**

- **When creating a workflow powered by Handoffs, we first need to create all the Agents that our workflow will use**

- **You are free to create multiple Agents using diverse tools, models and handoffs, and orchestrate your own workflow using these Agents**

- **Once all our Agents created, we update our previous defined Agents with a list of handoffs available.**

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)

Agents Handoffs

```python
import os
from mistralai import Mistral

api_key = os.environ["MISTRAL_API_KEY"]
client = Mistral(api_key)

# Create your agents
finance_agent = client.beta.agents.create(
    model="mistral-large-latest",
    description="Agent used to answer financial related requests",
    name="finance-agent",
)
web_search_agent = client.beta.agents.create(
    model="mistral-large-latest",
    description="Agent that can search online for any information if needed",
    name="websearch-agent",
    tools=[{"type": "web_search"}],
)
ecb_interest_rate_agent = client.beta.agents.create(
    model="mistral-large-latest",
    description="Can find the current interest rate of the European central bank",
    name="ecb-interest-rate-agent",
    tools=[...] # check previous pages for implementation

graph_agent = client.beta.agents.create(
    model="mistral-large-latest",
    name="graph-drawing-agent",
    description="Agent used to create graphs using the code interpreter tool.",
    instructions="Use the code interpreter tool when you have to draw a graph.",
    tools=[{"type": "code_interpreter"}]
)


# Define Agent Handoffs
# Allow the finance_agent to handoff the conversation to the ecb_interest_rate_agent or
web_search_agent
finance_agent = client.beta.agents.update(
    agent_id=finance_agent.id,
    handoffs=[ecb_interest_rate_agent.id, web_search_agent.id]
)

# Allow the ecb_interest_rate_agent to handoff the conversation to the graph_agent
ecb_interest_rate_agent = client.beta.agents.update(
    agent_id=ecb_interest_rate_agent.id,
    handoffs=[graph_agent.id]
)

# Allow the web_search_agent to handoff the conversation to the graph_agent
web_search_agent = client.beta.agents.update(
    agent_id=web_search_agent.id,
    handoffs=[graph_agent.id]
)
```

Source: Mistral Agents Docs

Compiled by: Dipanjan (DJ)