OpenAI assistants

> The Assistants API allows you to build AI assistants within your own applications. An Assistant has instructions and can leverage models, tools, and knowledge to respond to user queries. The Assistants API currently supports three types of tools: Code Interpreter, Retrieval, and Function calling

You can interact with OpenAI Assistants using OpenAI tools or custom tools. When using exclusively OpenAI tools, you can just invoke the assistant directly and get final answers. When using custom tools, you can run the assistant and tool execution loop using the built-in AgentExecutor or easily write your own executor.

Below we show the different ways to interact with Assistants. As a simple example, let's build a math tutor that can write and run code.

Using only OpenAI tools

```
from langchain.agents.openai_assistant import
OpenAIAssistantRunnable
```

```
interpreter_assistant =
OpenAIAssistantRunnable.create_assistant(
    name="langchain assistant",
    instructions="You are a personal math
tutor. Write and run code to answer math
questions.",
    tools=[{"type": "code_interpreter"}],
    model="gpt-4-1106-preview",
)
output =
interpreter_assistant.invoke({"content":
"What's 10 - 4 raised to the 2.7"})
output
```

```
[ThreadMessage(id='msg_qgxkD5kvkZyl0qOaL4czPFkZ
assistant_id='asst_0T8S7CJuUa4Y4hm1PF6n62v7',
content=
[MessageContentText(text=Text(annotations=[],
value='The result of the calculation \\(10 -
4^{2.7}\\) is approximately \\(-32.224\\).'),
type='text')], created_at=1700169519, file_ids=
[], metadata={}, object='thread.message',
role='assistant',
run_id='run_aH3ZgSWNk3vYIBQm3vpE8tr4',
thread_id='thread_9K6cYfx1RBh0pOWD8SxwVWW9')]
```

As a LangChain agent with arbitrary tools

Now let’s recreate this functionality using our own tools. For this example we’ll use the E2B sandbox runtime tool.

```
%pip install --upgrade --quiet  e2b
duckduckgo-search
```

```
import getpass

from langchain.tools import DuckDuckGoSearchRun
E2BDataAnalysisTool

tools =
[E2BDataAnalysisTool(api_key=getpass.getpass())
DuckDuckGoSearchRun()]
```

```
agent =
OpenAIAssistantRunnable.create_assistant(
    name="langchain assistant e2b tool",
    instructions="You are a personal math
tutor. Write and run code to answer math
questions. You can also search the
internet.",
    tools=tools,
    model="gpt-4-1106-preview",
    as_agent=True,
)
```

Using AgentExecutor

The OpenAIAssistantRunnable is compatible with the AgentExecutor, so we can pass it in as an agent directly to the executor. The AgentExecutor handles calling the invoked tools and uploading the tool outputs back to the Assistants API. Plus it comes with built-in LangSmith tracing.

```
from langchain.agents import AgentExecutor

agent_executor = AgentExecutor(agent=agent,
tools=tools)
agent_executor.invoke({"content": "What's the
weather in SF today divided by 2.7"})
```

```
{'content': "What's the weather in SF today
divided by 2.7",
 'output': "The search results indicate that
the weather in San Francisco is 67 °F. Now I
will divide this temperature by 2.7 and
provide you with the result. Please note that
this is a mathematical operation and does not
represent a meaningful physical
quantity.\n\nLet's calculate 67 °F divided by
2.7.\nThe result of dividing the current
temperature in San Francisco, which is 67 °F,
by 2.7 is approximately 24.815.",
 'thread_id':
'thread_hcpYI0tfpB9mHa9d95W7nK2B',
 'run_id': 'run_qOuVmPXS9xlV3XNPcfP8P9W2'}
```

LangSmith trace

Custom execution

Or with LCEL we can easily write our own execution loop for running the assistant. This gives us full control over execution.

```
agent =
OpenAIAssistantRunnable.create_assistant(
    name="langchain assistant e2b tool",
    instructions="You are a personal math
tutor. Write and run code to answer math
questions.",
    tools=tools,
    model="gpt-4-1106-preview",
    as_agent=True,
)
```

```
from langchain_core.agents import AgentFinish
```

```
def execute_agent(agent, tools, input):
    tool_map = {tool.name: tool for tool in
tools}
    response = agent.invoke(input)
    while not isinstance(response,
AgentFinish):
        tool_outputs = []
        for action in response:
            tool_output =
tool_map[action.tool].invoke(action.tool_input)
            print(action.tool,
action.tool_input, tool_output, end="\n\n")
            tool_outputs.append(
                {"output": tool_output,
"tool_call_id": action.tool_call_id}
            )
        response = agent.invoke(
            {
                "tool_outputs": tool_outputs,
                "run_id": action.run_id,
                "thread_id": action.thread_id,
            }
        )

    return response
```

```
response = execute_agent(agent, tools,
{"content": "What's 10 - 4 raised to the
2.7"})
print(response.return_values["output"])
```

```
e2b_data_analysis {'python_code': 'result =
10 - 4 ** 2.7\nprint(result)'} {"stdout":
"-32.22425314473263", "stderr": "",
"artifacts": []}

\( 10 - 4^{2.7} \) equals approximately
-32.224.
```

Using existing Thread

To use an existing thread we just need to pass the “thread_id” in when invoking the agent.

```
next_response = execute_agent(
    agent,
    tools,
    {"content": "now add 17.241",
"thread_id":
response.return_values["thread_id"]},
)
print(next_response.return_values["output"])
```

```
e2b_data_analysis {'python_code': 'result =
10 - 4 ** 2.7 + 17.241\nprint(result)'}
{"stdout": "-14.983253144732629", "stderr":
"", "artifacts": []}

\( 10 - 4^{2.7} + 17.241 \) equals
approximately -14.983.
```

Using existing Assistant

To use an existing Assistant we can initialize the `OpenAIAssistantRunnable` directly with an `assistant_id`.

```
agent =
OpenAIAssistantRunnable(assistant_id="
<ASSISTANT_ID>", as_agent=True)
```