A Comparison of Assistant and Non-Assistant Tool Calling Pipelines

Introduction

At a high level, the logic behind assistant tool calling and non-assistant tool calling is fundamentally the same: the model instructs the user to call specific function(s) in order to answer the user's query. The user then executes the function and returns the result to the model, which uses it to generate an answer. This process is identical for both.

However, since the assistant specifies the function definitions and access to tools as part of the Assistant configuration within the OpenAI or Azure OpenAI dashboard rather than within your pipelines, there will be major differences in the pipeline configuration. Additionally submitting tool responses to an Assistant comes with significant changes and challenges since the Assistant owns the conversational history rather than the pipeline.

This article focuses on contrasting these differences. For a detailed understanding of assistant pipelines and non-assistant pipelines, please refer to the following article:

Non-assistant pipelines: Introducing Tool Calling Snaps and LLM Agent Pipelines

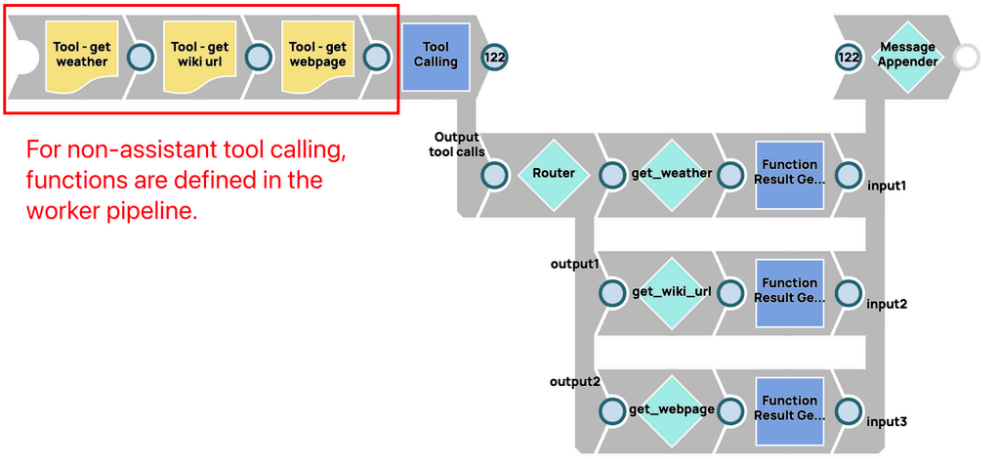
Assistant pipelines: Introducing Assistant Tool Calling Pipelines

Part 1: Which System to Use: Non-Assistant or Assistant?

When to Use Non-Assistant Tool Calling Pipelines:

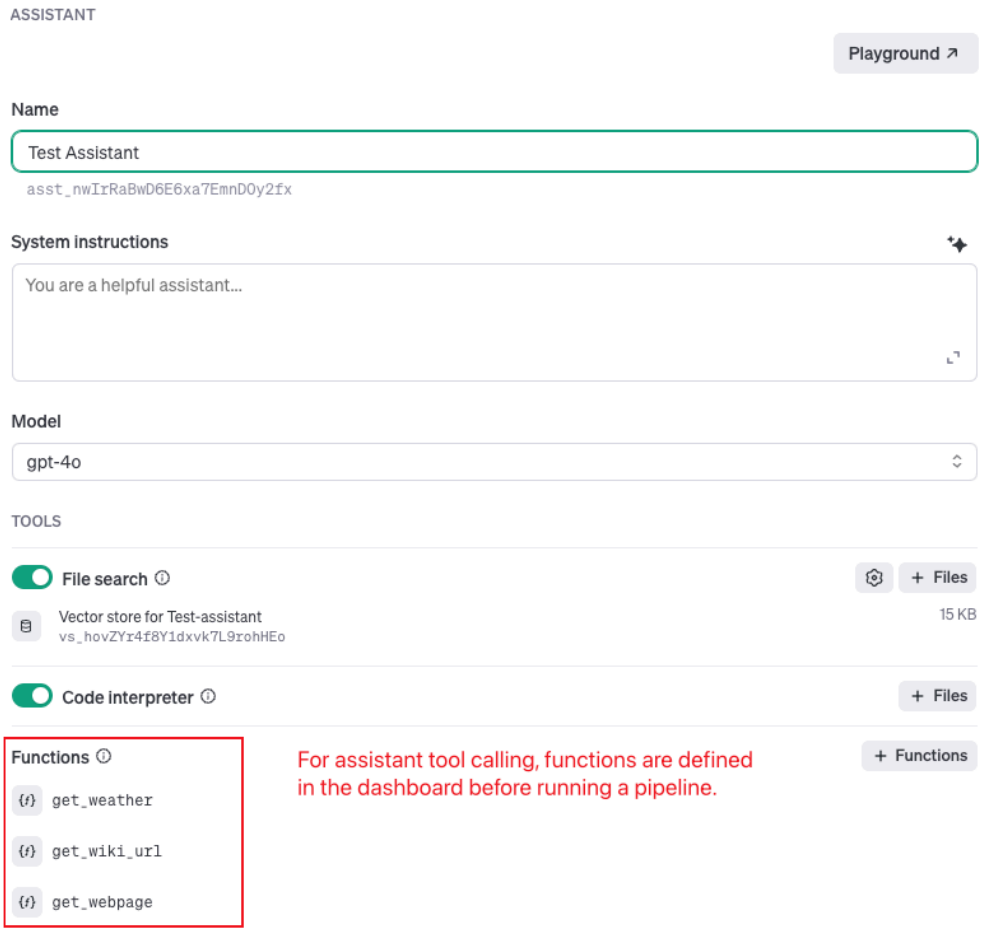
Non-Assistant Tool Calling Pipelines offer greater flexibility and control over the tool calling process, making them suitable for the following specific scenarios.

- **When preferring a "run-time" approach:** Non-Assistant pipelines exhibit greater flexibility in function definition, offering a more "runtime" approach. You can dynamically adjust the available functions by simply adding or removing Function Generator snaps within the pipeline.



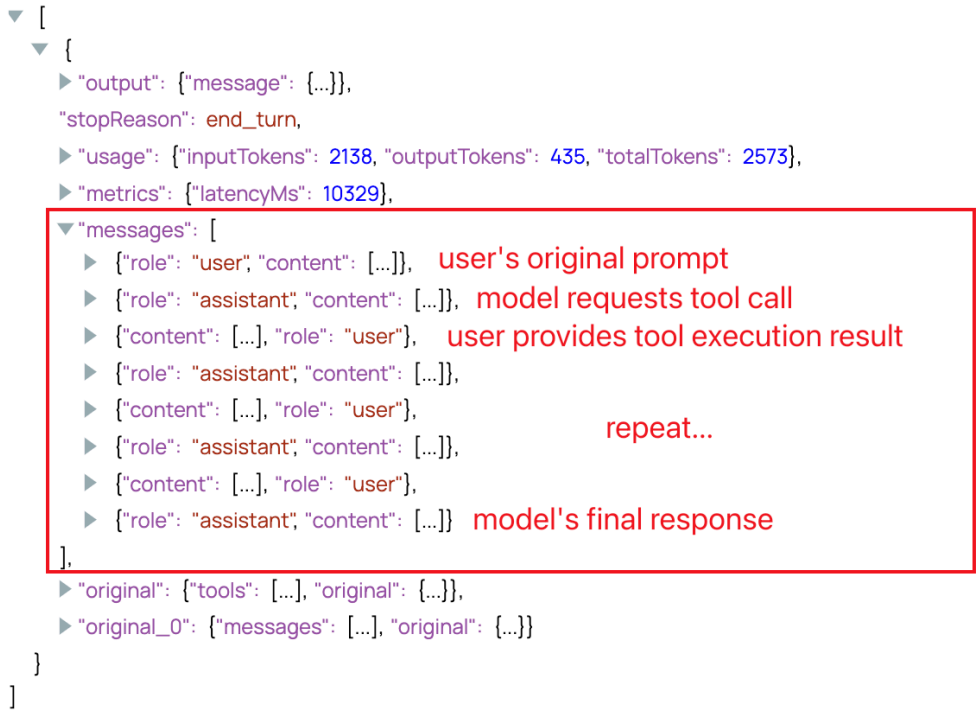
Non-assistant Tool Calling

In contrast, Assistant Tool Calling Pipelines necessitate a "design-time" approach. All available functions must be pre-defined within the Assistant configuration, requiring modifications to the Assistant definition in the OpenAI/Azure OpenAI dashboard.



Assistant Tool Calling

- **When wanting detailed chat history:** Non-Assistant pipelines provide a comprehensive history of the interaction between the model and the tools in the output message list. The message list within the Non-Assistant pipeline preserves every model response and the results of each function execution. This detailed logging allows for thorough debugging, analysis, and auditing of the tool calling process.

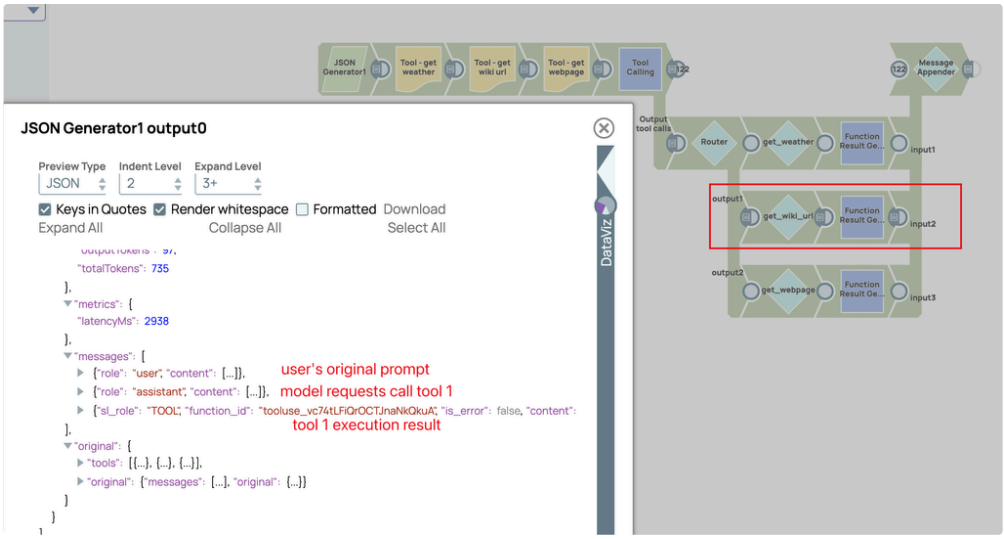


Non-assistant Tool Calling

In contrast, Assistant pipelines maintain a more concise message history, focusing on key steps and omitting some intermediate details. While this can simplify the overall view of the message list, it can also make it more difficult to trace the exact sequence of events or diagnose issues that may arise during tool execution in child pipelines.

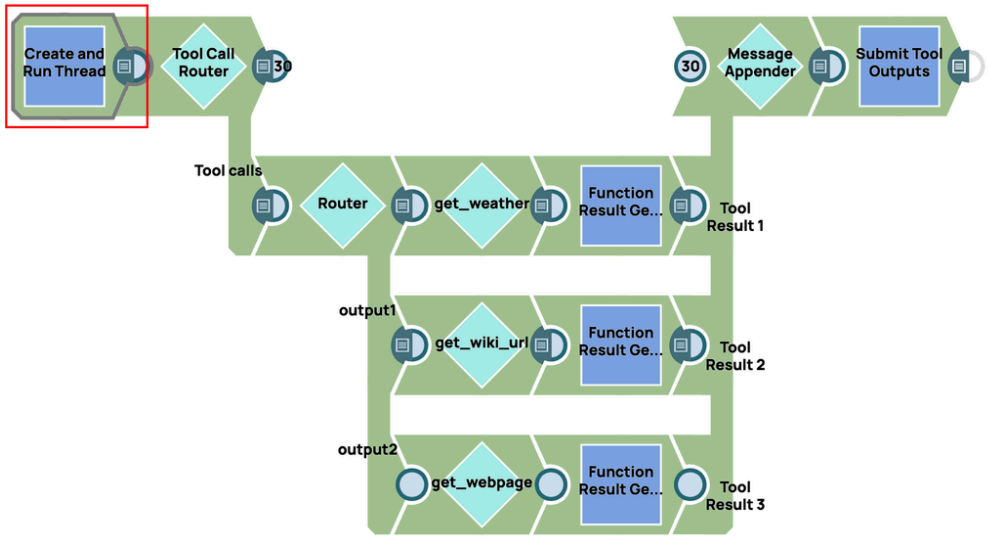
Assistant Tool Calling

- **When needing easier debugging and iterative development:** Non-Assistant pipelines facilitate more granular debugging and iterative development. You can easily simulate individual steps of the agent by making calls to the model with specific function call histories. This allows for more precise control and experimentation during development, enabling you to isolate and address issues more effectively. For example, by providing three messages, we can "force" the model to call the second tool, allowing us to inspect the tool calling process and its result against our expectations.



In contrast, debugging and iterating with Assistant pipelines can be more cumbersome. Since Assistants manage the conversation history internally, to simulate a specific step, you often need to replay the entire interaction from the beginning, potentially requiring multiple iterations to reach the desired state. This internal management of history makes it less straightforward to isolate and debug specific parts of the interaction.

To simulate calling the third tool, we need to start a new thread from scratch and then call tool1 and tool2, repeating the preceding process. The current thread cannot be reused.



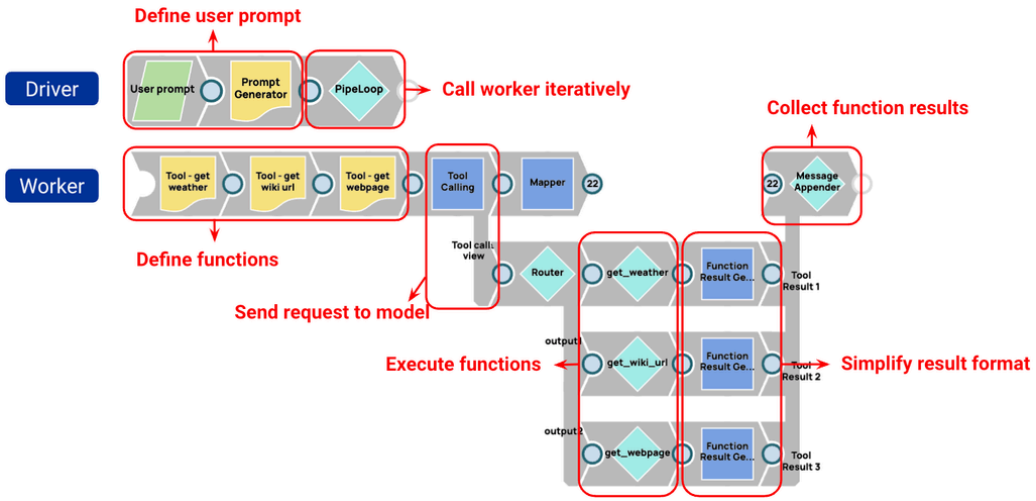
When to Use Assistant Tool Calling Pipelines:

Assistant Tool Calling Pipelines also offer a streamlined approach to integrating LLMs with external tools, prioritizing ease of use and built-in functionalities. Consider using Assistant pipelines in the following situations:

- **For simplified pipeline design**: Assistant pipelines reduce pipeline complexity by eliminating the need for Tool Generator snaps. In Non-Assistant pipelines, these snaps are essential for dynamically generating tool definitions within the pipeline itself. With Assistant pipelines, tool definitions are configured beforehand within the Assistant settings in the OpenAI/Azure OpenAI dashboard. This pre-configuration results in shorter, more manageable pipelines, simplifying development and maintenance.
- **When leveraging built-in tools is required**: If your use case requires functionalities like searching external files or executing code, Assistant pipelines offer these capabilities out-of-the-box through their built-in File Search and Code Interpreter tools (see Part 5 for more details). These tools provide a convenient and efficient way to extend the LLM's capabilities without requiring custom implementation within the pipeline.

Part 2: A brief introduction to two pipelines

Non-assistant tool calling pipelines

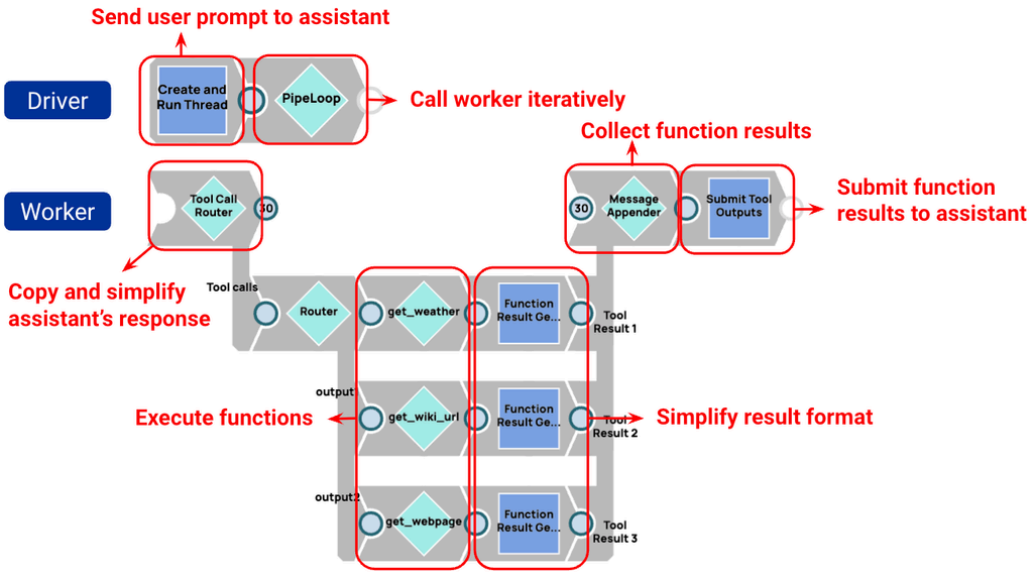


Non-assistant tool calling pipelines

Key points:

- Functions are defined in the worker.
- The worker pipeline's Tool Calling snap manages all model interactions.
- Function results are collected and sent to the model in the next iteration via the Tool Calling snap.

Assistant tool calling pipelines



Assistant tool calling pipelines

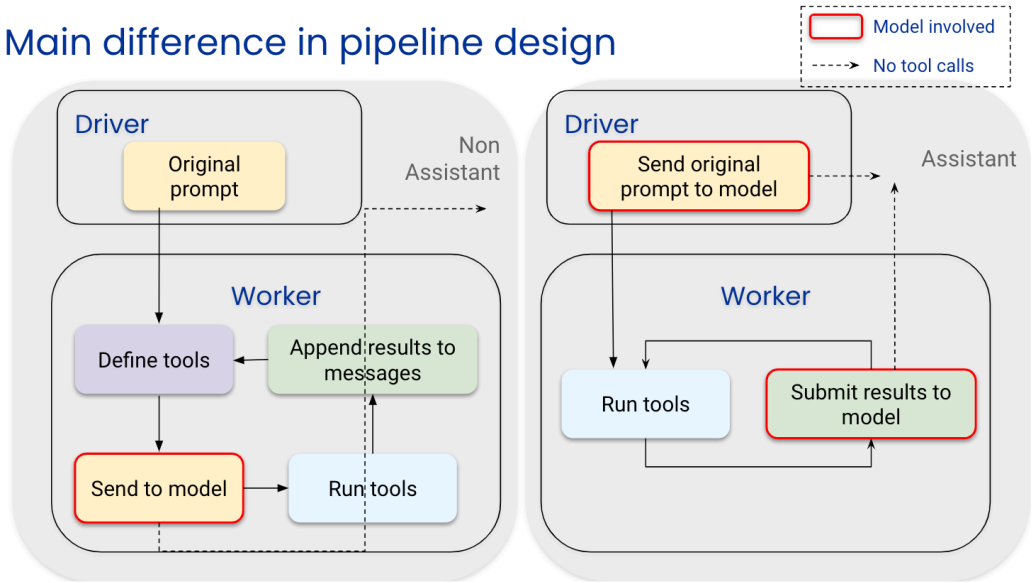
Key points:

- No need to define functions in any pipeline. Functions are pre-defined in the assistant.
- Two snaps : interact with the model: **Create and Run Thread**, and **Submit Tool Outputs**.
- Function results are collected and sent to the model immediately during the current iteration.

Part 3: Comparison between two pipelines

Here are two primary reasons why the assistant and non-assistant pipelines differ, listed in decreasing order of importance:

1. **Distinct methods of submitting tool results:**
   - For non-assistant pipelines, tool results are appended to the message history list and subsequently forwarded to the model during the **next** iteration.
     Non-assistant pipelines exhibit a "**while-loop**" behavior, where the worker interacts with the model at the beginning of the iteration, and while any tools need to be called, the worker executes those tool(s).
   - In contrast, for assistants, tool results are specifically sent to a dedicated endpoint designed to handle tool call results within the **current** iteration.
     The assistant pipelines operate more like a "**do-while-loop**." The driver initiates the interaction by sending the prompt to the model. Subsequently, the worker execute the tool(s) first and interacts with the model at the end of the iteration to deliver tool results.
2. **Predefined and stored tool definitions for assistants:**
   - Unlike non-assistant pipelines, assistants have the capability to predefine and store function definitions. This eliminates the need for the three `Function Generator` snaps to repeatedly transmit tool definitions to the model with each request. Consequently, the worker pipeline for assistants appears shorter.



Due to the aforementioned differences, non-assistant pipelines have only one interaction point with the model, located in the worker.

In contrast, assistant pipelines involve two interaction points: the driver sends the initial prompt to the model, while the worker sends tool results back to the model.

Part 4: Differences in snap settings

Stop condition of Pipeloop

A key difference in snap settings lies in the stop condition of the pipeloop.

> Assistant pipeline's stop condition: `$run.required_action == null`.
>
> Non-assistant pipeline's stop condition: `$finish_reason != "tool_calls"`.

Assistant's output

```
"required_action": {
  "type": "submit_tool_outputs",
  "submit_tool_outputs": {
    "tool_calls": [
      {
        "id": "call_xj74QONQ9XhpLSGMyhd0WLtV",
        "type": "function",
        "function": {
          "name": "get_webpage",
          "arguments": "{"url":"https://en.wikipedia.org/wiki/San_Francisco,_CA"}"
        }
      }
    ]
  }
},
```


Example when tool calls are required


```
"cancelled_at": null,
"failed_at": null,
"completed_at": 1731345620,
"required_action": null,
"last_error": null,
"model": "gpt-4o",
"instructions": "",
"tools": [{...}, {...}, {...}, {...}, {...}],
"tool_resources": {},
"metadata": {},
```


Example when tool calls are NOT required

Non-assistant's output


```
"choices": [
  {
    "index": 0,
    "message": {
      "role": "assistant",
      "content": null,
      "tool_calls": [
        {"id": "call_lWfxGMVOAS2cgA9bb5aWsQNL", "type": "function", "function": {...}},
        {"id": "call_AkOuBx6mGuEPu0EH1yy7sY4j", "type": "function", "function": {...}}
      ],
      "refusal": null
    },
    "logprobs": null,
    "finish_reason": "tool_calls"
  }
```


Example when tool calls are required


```
"choices": [
  {
    "index": 0,
    "message": {"role": "assistar
    "logprobs": null,
    "finish_reason": "stop"
  }
```

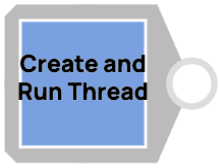

Example when tool calls are NOT required

Part 5: Assistant's two built-in tools

The assistant not only supports all functions that can be defined in non-assistant pipelines but also provides two special built-in functions, **file search** and **code interpreter**, for user convenience.

If the model determines that either of these tools is required, it will automatically call and execute the tool within the assistant without requiring manual user intervention.

You don't need a tool call pipeline to experiment with file search and code interpreter. A simple create and run thread snap is sufficient.



File search

> File Search augments the Assistant with knowledge from outside its model, such as proprietary product information or documents provided by your users. OpenAI automatically parses and chunks your documents, creates and stores the embeddings, and use both vector and keyword search to retrieve relevant content to answer user queries.

Example

Prompt: What is the number of federal fires between 2018 and 2022?

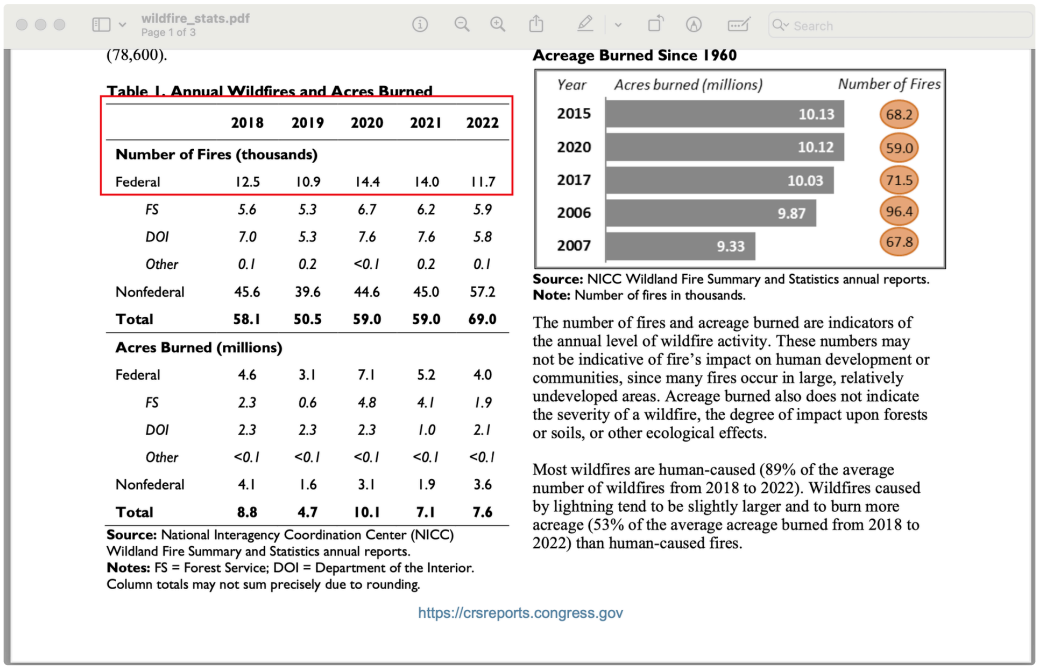
The assistant's response is as below:

Assistant's response

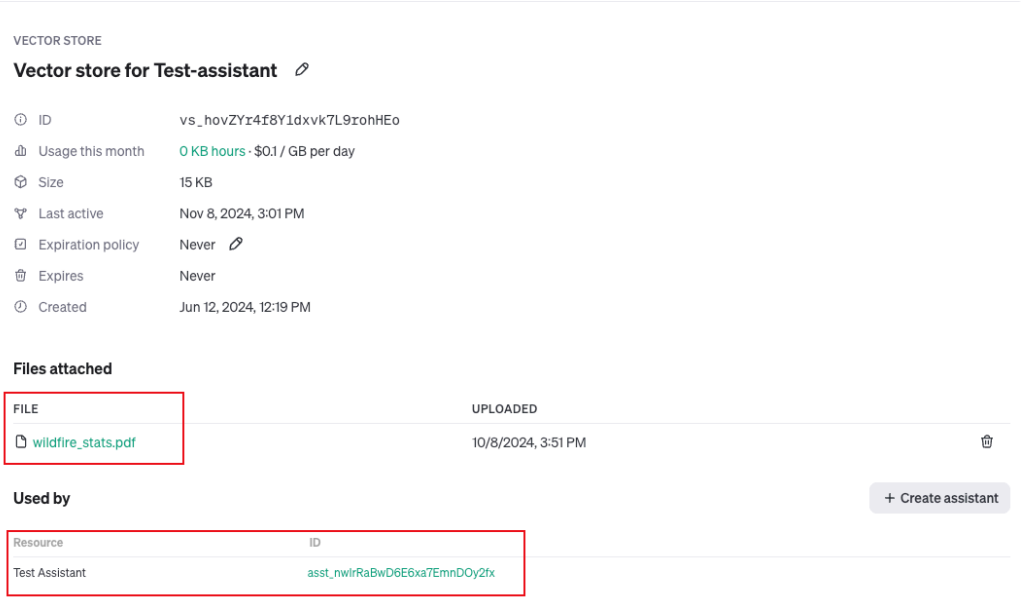

```
[
  {
    "messages": [
      {
        "id": "msg_cyvIQG7htmHnwTTbfkrES3ms",
        "object": "thread.message",
        "created_at": 1731106910,
        "assistant_id": "asst_nwIrRaBwD6E6xa7EmnDOy2fx",
        "thread_id": "thread_ciR3mFR1jEcXK07pX06jCRgM",
        "run_id": "run_61Xt9zvpXLYxfgIfF7GV8Nz2",
        "role": "assistant",
        "content": [
          {
            "type": "text",
            "text": {
              "value": "The number of federal fires between 2018 and 2022 is as follows:\n\n- 2018: 12,500\n- 2019: 10,900\n- 2020: 14,400\n- 2021: 14,000\n- 2022: 11,700【4:1†wildfire_stats.pdf】.",
              "annotations": [
                {
                  "type": "file_citation",
                  "text": "【4:1†wildfire_stats.pdf】",
                  "start_index": 140,
                  "end_index": 164,
                  "file_citation": {
                    "file_id": "file-fJGINZ4R7XlIGtjfvv0W71CH"
                  }
```

```
              }
            ]
          }
        }
      ],
      "attachments": [],
      "metadata": {}
    },
    {
      "id": "msg_LBP4fengd7GlQnu7ZkfqvM2W",
      "object": "thread.message",
      "created_at": 1731106907,
      "assistant_id": null,
      "thread_id": "thread_ciR3mFR1jEcXK07pX06jCRgM",
      "run_id": null,
      "role": "user",
      "content": [
        {
          "type": "text",
          "text": {
            "value": "What is the number of federal fires between 2018 and 2022",
            "annotations": []
          }
        }
      ],
      "attachments": [],
      "metadata": {}
    }
  ],
  "run": {...}
}
]
```


The assistant's response is correct. As the answer to the prompt is in the first row of a table on the first page of wildfire_stats.pdf , a document accessible to the assistant via a vector store.



Answer to the prompt



The file is from a vector store used by the assistant.

Code Interpreter

> Code Interpreter allows Assistants to write and run Python code in a sandboxed execution environment. This tool can process files with diverse data and formatting, and generate files with data and images of graphs. Code Interpreter allows your Assistant to run code iteratively to solve challenging code and math problems. When your Assistant writes code that fails to run, it can iterate on this code by attempting to run different code until the code execution succeeds.

Example

Prompt: Find the number of federal fires between 2018 and 2022 and use Matplotlib to draw a line chart.

* Matplotlib is a python library for creating plots.

The assistant's response is as below:

Assistant's response


```
[
  {
    "messages": [
      {
        "id": "msg_lzBiM0J4sC0Zji510f1NOjjM",
        "object": "thread.message",
        "created_at": 1731108369,
        "assistant_id": "asst_nwIrRaBwD6E6xa7EmnDOy2fx",
        "thread_id": "thread_3q9AV6ivrYqqzsexv1rzMFSV",
        "run_id": "run_DbjZQbBVQgoVge74PRbyGh44",
        "role": "assistant",
        "content": [
          {
            "type": "image_file",
            "image_file": {
              "file_id": "file-CLHOiYRfuWD45DsN6M4b8ga9"
            }
          },
          {
            "type": "text",
```

```
            "text": {
              "value": "Here is the line chart showing the number of federal fires from 2018 to 2022. As you can see, there is a fluctuation in the number of fires over these years, with a peak in 2020.",
              "annotations": []
            }
          }
        ],
        "attachments": [],
        "metadata": {}
      },
      {
       ...,
        "content": [
          {
            "type": "text",
            "text": {
              "value": "The number of federal fires between 2018 and 2022 was as follows:\n\n- 2018: 12.5 thousand fires\n- 2019: 10.9 thousand fires\n- 2020: 14.4 thousand fires\n- 2021: 14.0 thousand fires\n- 2022: 11.7 thousand fires【4:0†wildfire_stats.pdf】.\n\nI will now create a line chart using Matplotlib to represent this data.",
              "annotations": [
                {
                  "type": "file_citation",
                  "text": "【4:0†wildfire_stats.pdf】",
                  "start_index": 206,
                  "end_index": 230,
                  "file_citation": {
                    "file_id": "file-fJGINZ4R7XlIGtjfvv0W71CH"
                  }
                }
              ]
            }
          }
        ],
        ...
      },
      {
        ...,
        "content": [
          {
            "type": "text",
            "text": {
              "value": "Find the number of federal fires between 2018 and 2022 and use Matplotlib to draw a line chart.",
              "annotations": []
            }
          }
        ],
        ...
      }
    ],
    "run": {...}
  }
]
```


From the response, we can see that the assistant indicated it used file search to find 5 years of data and then generated an image file. This file can be downloaded from the assistant's dashboard under storage-files. Simply add a file extension like .png to see the image.

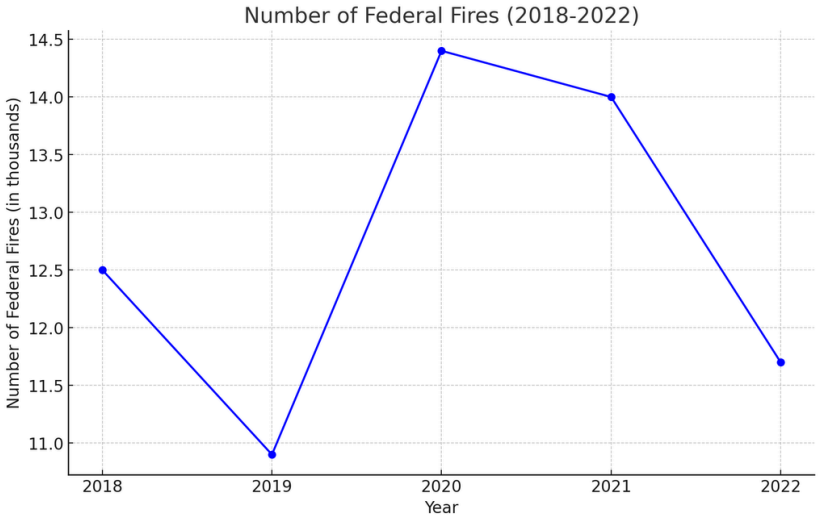


Image file generated by assistant

Part 6: Key Differences Summarized

Feature	Non-Assistant Tool Calling Pipelines	Assistant Tool Calling Pipelines
Function Definition	Defined within the worker pipeline using Function Generator snaps.	Pre-defined and stored within the Assistant configuration in the OpenAI/Azure OpenAI dashboard.
Tool Result Submission	Appended to the message history and sent to the model in the *next* iteration.	Sent to a dedicated endpoint within the *current* iteration.
Model Interaction Points	One (in the worker pipeline).	Two (driver sends initial prompt, worker sends tool results).
Built-in Tools	None.	File Search and Code Interpreter.
Pipeline Complexity	More complex pipeline structure due to function definition within the pipeline.	Simpler pipeline structure as functions are defined externally.