Documentation API reference Forum Help

Streaming is now available in the Assistants API.

Learn more Dismiss

Tools

Beta

Give Assistants access to OpenAI-hosted tools like Code Interpreter and Knowledge Retrieval, or build your own tools using Function calling. Usage of OpenAI-hosted tools comes at an additional fee — visit our help center article to learn more about how these tools are priced.

The Assistants API is in **beta** and we are actively working on adding more functionality. Share your feedback in our Developer Forum!

Code Interpreter

Code Interpreter allows the Assistants API 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.

Code Interpreter is charged at $0.03 per session. If your Assistant calls Code Interpreter simultaneously in two different threads (e.g., one thread per end-user), two Code Interpreter sessions are created. Each session is active by default for one hour, which means that you only pay for one session per if users interact with Code Interpreter in the same thread for up to one hour.

Enabling Code Interpreter

Pass the `code_interpreter` in the `tools` parameter of the Assistant object to enable Code Interpreter:

python

```python
assistant = client.beta.assistants.create(
  instructions="You are a personal math tutor. When asked a math questi
  model="gpt-4-turbo-preview",
  tools=[{"type": "code_interpreter"}]
)
```

The model then decides when to invoke Code Interpreter in a Run based on the nature of the user request. This behavior can be promoted by prompting in the Assistant's `instructions` (e.g., “write code to solve this problem”).

Passing files to Code Interpreter

Code Interpreter can parse data from files. This is useful when you want to provide a large volume of data to the Assistant or allow your users to upload their own files for analysis. Note that files uploaded for Code Interpreter are not indexed for retrieval. See the Knowledge Retrieval section below for more details on indexing files for retrieval.

Files that are passed at the Assistant level are accessible by all Runs with this Assistant:

python

```python
# Upload a file with an "assistants" purpose
file = client.files.create(
  file=open("speech.py", "rb"),
  purpose='assistants'
)

# Create an assistant using the file ID
assistant = client.beta.assistants.create(
  instructions="You are a personal math tutor. When asked a math quest
  model="gpt-4-turbo-preview",
  tools=[{"type": "code_interpreter"}],
  file_ids=[file.id]
)
```

> You do not pay for files attached to an Assistant or Message when used with Code Interpreter. You are only charged for files that are indexed for retrieval which happens automatically if the Retrieval tool is enabled.

Files can also be passed at the Thread level. These files are only accessible in the specific Thread. Upload the File using the File upload endpoint and then pass the specific Thread. Upload the File using the File upload endpoint and then pass the File ID as part of the Message creation request:

python

```python
thread = client.beta.threads.create(
  messages=[
    {
      "role": "user",
      "content": "I need to solve the equation `3x + 11 = 14`. Can you
      "file_ids": [file.id]
    }
  ]
)
```

Files have a maximum size of 512 MB. Code Interpreter supports a variety of file formats including `.csv`, `.pdf`, `.json` and many more. More details on the file extensions (and their corresponding MIME-types) supported can be found in the Supported files section below.

Reading images and files generated by Code Interpreter

Code Interpreter in the API also outputs files, such as generating image diagrams, CSVs, and PDFs. There are two types of files that are generated:

1. Images
2. Data files (e.g. a `csv` file with data generated by the Assistant)

When Code Interpreter generates an image, you can look up and download this file in the `file_id` field of the Assistant Message response:

```
{
    "id": "msg_abc123",
    "object": "thread.message",
    "created_at": 1698964262,
    "thread_id": "thread_abc123",
    "role": "assistant",
    "content": [
    {
      "type": "image_file",
      "image_file": {
        "file_id": "file-abc123"
      }
    }
```

```
  ]
  # ...
}
```

The file content can then be downloaded by passing the file ID to the Files API:

python

```python
from openai import OpenAI

client = OpenAI()

image_data = client.files.content("file-abc123")
image_data_bytes = image_data.read()

with open("./my-image.png", "wb") as file:
    file.write(image_data_bytes)
```

When Code Interpreter references a file path (e.g., ”Download this csv file”), file paths are listed as annotations. You can convert these annotations into links to download the file:

```
{
  "id": "msg_abc123",
  "object": "thread.message",
  "created_at": 1699073585,
  "thread_id": "thread_abc123",
  "role": "assistant",
  "content": [
    {
      "type": "text",
      "text": {
        "value": "The rows of the CSV file have been shuffled
        "annotations": [
          {
            "type": "file_path",
            "text": "sandbox:/mnt/data/shuffled_file.csv",
            "start_index": 167,
            "end_index": 202,
            "file_path": {
              "file_id": "file-abc123"
            }
          }
          ...
```

Input and output logs of Code Interpreter

By listing the steps of a Run that called Code Interpreter, you can inspect the code `input` and `outputs` logs of Code Interpreter:

python

```python
run_steps = client.beta.threads.runs.steps.list(
  thread_id=thread.id,
  run_id=run.id
)
```

```
{
  "object": "list",
  "data": [
    {
      "id": "step_abc123",
      "object": "thread.run.step",
      "type": "tool_calls",
      "run_id": "run_abc123",
      "thread_id": "thread_abc123",
      "status": "completed",
      "step_details": {
        "type": "tool_calls",
        "tool_calls": [
          {
            "type": "code",
            "code": {
              "input": "# Calculating 2 + 2\nresult = 2 + 2\r
              "outputs": [
                {
                  "type": "logs",
                  "logs": "4"
                }
                        ...
 }
```

Knowledge Retrieval

Retrieval augments the Assistant with knowledge from outside its model, such as proprietary product information or documents provided by your users. Once a file is uploaded and passed to the Assistant, OpenAI will automatically chunk your documents, index and store the embeddings, and implement vector search to retrieve relevant content to answer user queries.

Enabling Retrieval

Pass the `retrieval` in the `tools` parameter of the Assistant to enable Retrieval:

python

```python
assistant = client.beta.assistants.create(
  instructions="You are a customer support chatbot. Use your knowledge
  model="gpt-4-turbo-preview",
  tools=[{"type": "retrieval"}]
)
```

> If you enable retrieval for a specific Assistant, all the files attached will be automatically indexed and you will be charged the $0.20/GB per assistant per day. You can enabled/disable retrieval by using the Modify Assistant endpoint.

How it works

The model then decides when to retrieve content based on the user Messages. The Assistants API automatically chooses between two retrieval techniques:

1. it either passes the file content in the prompt for short documents, or
2. performs a vector search for longer documents

Retrieval currently optimizes for quality by adding all relevant content to the context of model calls. We plan to introduce other retrieval strategies to enable developers to choose a different tradeoff between retrieval quality and model usage cost.

Uploading files for retrieval

Similar to Code Interpreter, files can be passed at the Assistant-level or individual Message-level.

python

```python
# Upload a file with an "assistants" purpose
file = client.files.create(
  file=open("knowledge.pdf", "rb"),
  purpose='assistants'
)

# Add the file to the assistant
assistant = client.beta.assistants.create(
  instructions="You are a customer support chatbot. Use your knowledge
  model="gpt-4-turbo-preview",
  tools=[{"type": "retrieval"}],
  file_ids=[file.id]
)
```

When a file is attached at the Message-level, it is only accessible within the specific Thread the Message is attached to. After having uploaded a file, you can pass the ID of this File when creating the Message. Note that you are not charged based on the size of the files you upload via the Files API but rather based on which files you attach to a specific Assistant or Message that get indexed.

python

```python
message = client.beta.threads.messages.create(
  thread_id=thread.id,
  role="user",
  content="I can not find in the PDF manual how to turn off this device
  file_ids=[file.id]
)
```

The maximum file size is 512 MB and no more than 2,000,000 tokens (computed automatically when you attach a file). Retrieval supports a variety of file formats including `.pdf`, `.md`, `.docx` and many more. More details on the file extensions (and their corresponding MIME-types) supported can be found in the Supported files section below.

Retrieval pricing

Retrieval is priced at $0.20/GB per assistant per day. Attaching a single file ID to multiple assistants will incur the per assistant per day charge when the retrieval tool is enabled. For example, if you attach the same 1 GB file to two different Assistants with the retrieval tool enabled (e.g., customer-facing Assistant #1 and internal employee Assistant #2), you'll be charged twice for this storage fee (2 * $0.20 per employee Assistant #2), you'll be charged twice for this storage fee (2 * $0.20 per day). This fee does not vary with the number of end users and threads retrieving knowledge from a given assistant.

In addition, files attached to messages are charged on a per-assistant basis if the messages are part of a run where the retrieval tool is enabled. For example, running an assistant with retrieval enabled on a thread with 10 messages each with 1 unique file (10 total unique files) will incur a per-GB per-day charge on all 10 files (in addition to any files attached to the assistant itself).

Deleting files

To remove a file from the assistant, you can detach the file from the assistant:

python

```python
file_deletion_status = client.beta.assistants.files.delete(
  assistant_id=assistant.id,
  file_id=file.id
)
```

Detaching the file from the assistant removes the file from the retrieval index and means you will no longer be charged for the storage of the indexed file.

File citations

When Code Interpreter outputs file paths in a Message, you can convert them to corresponding file downloads using the `annotations` field. See the Annotations section for an example of how to do this.

```json
{
    "id": "msg_abc123",
    "object": "thread.message",
    "created_at": 1699073585,
    "thread_id": "thread_abc123",
    "role": "assistant",
    "content": [
      {
        "type": "text",
        "text": {
          "value": "The rows of the CSV file have been shuffl
          "annotations": [
            {
              "type": "file_path",
              "text": "sandbox:/mnt/data/shuffled_file.csv",
              "start_index": 167,
              "end_index": 202,
              "file_path": {
                "file_id": "file-abc123"
              }
            }
          ]
        }
      }
    ],
    "file_ids": [
      "file-abc456"
    ],
        ...
  },
```

Function calling

Similar to the Chat Completions API, the Assistants API supports function calling. Function calling allows you to describe functions to the Assistants and have it intelligently return the functions that need to be called along with their arguments. The Assistants API will pause execution during a Run when it invokes functions, and you can supply the results of the function call back to continue the Run execution.

Defining functions

First, define your functions when creating an Assistant:

python

```
assistant = client.beta.assistants.create(
  instructions="You are a weather bot. Use the provided functions to a
  model="gpt-4-turbo-preview",
  tools=[{
      "type": "function",
    "function": {
      "name": "getCurrentWeather",
      "description": "Get the weather in location",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {"type": "string", "description": "The city and
          "unit": {"type": "string", "enum": ["c", "f"]}
        },
        "required": ["location"]
      }
    }
  }, {
    "type": "function",
    "function": {
      "name": "getNickname",
      "description": "Get the nickname of a city",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {"type": "string", "description": "The city and
        },
        "required": ["location"]
      }
    }
  }]
)
```

Reading the functions called by the Assistant

When you initiate a Run with a user Message that triggers the function, the Run will enter a `pending` status. After it processes, the run will enter a `requires_action` state which you can verify by retrieving the Run. The model can provide multiple functions to call at once using parallel function calling:

```
{
  "id": "run_abc123",
  "object": "thread.run",
  "assistant_id": "asst_abc123",
  "thread_id": "thread_abc123",
  "status": "requires_action",
  "required_action": {
    "type": "submit_tool_outputs",
    "submit_tool_outputs": {
      "tool_calls": [
        {
          "id": "call_abc123",
          "type": "function",
          "function": {
            "name": "getCurrentWeather",
            "arguments": "{\"location\":\"San Francisco\"}"
          }
        },
        {
          "id": "call_abc456",
          "type": "function",
          "function": {
            "name": "getNickname",
            "arguments": "{\"location\":\"Los Angeles\"}"
          }
        }
      ]
    }
  },
...
```

Submitting functions outputs

You can then complete the Run by submitting the tool output from the function(s) you call. Pass the `tool_call_id` referenced in the `required_action` object above to match output to each function call.

python

```
run = client.beta.threads.runs.submit_tool_outputs(
  thread_id=thread.id,
  run_id=run.id,
  tool_outputs=[
      {
        "tool_call_id": call_ids[0],
        "output": "22C",
      },
      {
        "tool_call_id": call_ids[1],
        "output": "LA",
      },
    ]
)
```

After submitting outputs, the run will enter the `queued` state before it continues it’s execution.

Supported files

For `text/` MIME types, the encoding must be one of `utf-8`, `utf-16`, or `ascii`.

FILE FORMAT	MIME TYPE	CODE INTERPRETER	RE
.c	text/x-c		
.cs	text/x-csharp		
.cpp	text/x-c++		
.csv	application/csv		
.doc	application/msword		
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.document		
.html	text/html		
.java	text/x-java		
.json	application/json		
.md	text/markdown		
.pdf	application/pdf		
.php	text/x-php		
.pptx	application/vnd.openxmlformats-officedocument.presentationml.presentation		
.py	text/x-python		
.py	text/x-script.python		
.rb	text/x-ruby		
.tex	text/x-tex		
.txt	text/plain		
.css	text/css		
.jpeg	image/jpeg		
.jpg	image/jpeg		
.js	text/javascript		

.js	text/javascript
.gif	image/gif
.png	image/png
.sh	application/x-sh
.tar	application/x-tar
.ts	application/typescript
.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
.xml	application/xml or "text/xml"
.zip	application/zip

Was this page useful?