OpenAI Platform

Code Interpreter   ⊡ Copy page

Allow models to write and run
Python to solve problems.

Overview

Containers

Work with files

Usage notes

The Code Interpreter tool allows models to write
and run Python code in a sandboxed environment
to solve complex problems in domains like data
analysis, coding, and math. Use it for:

  Processing files with diverse data and
  formatting

  Generating files with data and images of
  graphs

  Writing and running code iteratively to solve
  problems—for example, a model that writes
  code that fails to run can keep rewriting and
  running that code until it succeeds

  Boosting visual intelligence in our latest
  reasoning models (like o3 and o4-mini). The
  model can use this tool to crop, zoom, rotate,
  and otherwise process and transform images.

Here's an example of calling the Responses API with
a tool call to Code Interpreter:

```python
Use the Responses API with C…        python ↕    ⧉

1   from openai import OpenAI
2
3   client = OpenAI()
4
5   instructions = """
6   You are a personal math tutor. When ask
7   write and run code using the python too
8   """
9
10  resp = client.responses.create(
11      model="gpt-4.1",
12      tools=[
13          {
14              "type": "code_interpreter",
15              "container": {"type": "auto
16          }
17      ],
18      instructions=instructions,
19      input="I need to solve the equation
20  )
21
22  print(resp.output)
```

ⓘ While we call this tool Code Interpreter,
   the model knows it as the "python tool".
   Models usually understand prompts that
   refer to the code interpreter tool, however,
   the most explicit way to invoke this tool is
   to ask for "the python tool" in your prompts.

Containers

The Code Interpreter tool requires a
container object. A container is a fully sandboxed
virtual machine that the model can run Python code
in. This container can contain files that you upload,

or that it generates.

There are two ways to create containers:

1   Auto mode: as seen in the example above, you
    can do this by passing the
    `"container": { "type": "auto", files: ["file-1", "file-2"] }`
    property in the tool configuration while creating
    a new Response object. This automatically
    creates a new container, or reuses an active
    container that was used by a previous
    `code_interpreter_call` item in the model's
    context. Look for the
    `code_interpreter_call` item in the output
    of this API request to find the `container_id`
    that was generated or used.

2   Explicit mode: here, you explicitly
    create a container using the `v1/containers`
    endpoint, and assign its `id` as the
    `container` value in the tool configuration in
    the Response object. For example:

```python
Use explicit container creat…        python

 1   from openai import OpenAI
 2   client = OpenAI()
 3
 4   container = client.containers.create(na
 5
 6   response = client.responses.create(
 7       model="gpt-4.1",
 8       tools=[{
 9           "type": "code_interpreter",
10           "container": container.id
11       }],
12       tool_choice="required",
13       input="use the python tool to calcu
14   )
15
16   print(response.output_text)
```

Note that containers created with the auto mode are also accessible using the `/v1/containers` endpoint.

Expiration

We highly recommend you treat containers as ephemeral and store all data related to the use of this tool on your own systems. Expiration details:

> A container expires if it is not used for 20 minutes. When this happens, using the container in `v1/responses` will fail. You'll still be able to see a snapshot of the container's metadata at its expiry, but all data associated with the container will be discarded from our systems and not recoverable. You should download any files you may need from the container while it is active.

You can't move a container from an expired state to an active one. Instead, create a new container and upload files again. Note that any state in the old container's memory (like python objects) will be lost.

Any container operation, like retrieving the container, or adding or deleting files from the container, will automatically refresh the container's `last_active_at` time.

Work with files

When running Code Interpreter, the model can create its own files. For example, if you ask it to construct a plot, or create a CSV, it creates these images directly on your container. When it does so, it cites these files in the `annotations` of its next message. Here's an example:

```
1    {
2      "id": "msg_682d514e268c8191a89c38ea31
3      "content": [
4        {
5          "annotations": [
6            {
7              "file_id": "cfile_682d514b2e0
8              "index": null,
9              "type": "container_file_citat
10             "container_id": "cntr_682d513
11             "end_index": 0,
12             "filename": "cfile_682d514b2e
13             "start_index": 0
14           }
15         ],
16         "text": "Here is the histogram of
17         "type": "output_text",
18         "logprobs": []
19       }
20     ],
21     "role": "assistant",
22     "status": "completed",
23     "type": "message"
24   }
```

You can download these constructed files by calling the get container file content method.

Any files in the model input get automatically uploaded to the container. You do not have to explicitly upload it to the container.

Uploading and downloading files

Add new files to your container using Create container file. This endpoint accepts either a multipart upload or a JSON body with a `file_id`. List existing container files with List container files

and download bytes from
[Retrieve container file content](#).

Dealing with citations

Files and images generated by the model are
returned as annotations on the assistant's message.
`container_file_citation` annotations point to
files created in the container. They include the
`container_id` , `file_id` , and `filename` . You
can parse these annotations to surface download
links or otherwise process the files.

Supported files

FILE FORMAT	MIME TYPE
.c	text/x-c
.cs	text/x-csharp
.cpp	text/x-c++
.csv	text/csv
.doc	application/msword
.docx	application/vnd.openxmlformats-officedocument.wordprocessingml.documen
.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.presentat
.py	text/x-python
.py	text/x-script.python
.rb	text/x-ruby
.tex	text/x-tex
.txt	text/plain
.css	text/css
.js	text/javascript
.sh	application/x-sh
.ts	application/typescript
.csv	application/csv
.jpeg	image/jpeg
.jpg	image/jpeg
.gif	image/gif
.pkl	application/octet-stream
.png	image/png
.tar	application/x-tar
.xlsx	application/vnd.openxmlformats-officedocument.spreadsheetml.sheet
.xml	application/xml or "text/xml"
.zip	application/zip

Usage notes

API AVAILABILITY	RATE LIMITS	NOTES
✓ Responses ⊗ Chat Completions ✓ Assistants	100 RPM per org	Pricing ZDR and data residency