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.

Enabling Code Interpreter

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

node.js Copy

```
const assistant = await openai.beta.assistants.create({
  instructions: "You are a personal math tutor. When asked a math question, write and run co
  model: "gpt-4-1106-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.

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

node.js Copy

```
// Upload a file with an "assistants" purpose
const file = await openai.files.create({
  file: fs.createReadStream("mydata.csv"),
  purpose: "assistants",
});

// Create an assistant using the file ID
const assistant = await openai.beta.assistants.create({
  instructions: "You are a personal math tutor. When asked a math question, write and run c
  model: "gpt-4-1106-preview",
  tools: [{"type": "code_interpreter"}],
  file_ids: [file.id]
});
```

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 File ID as part of the Message creation request:

node.js Copy

```
const thread = await openai.beta.threads.create({
  messages: [
    {
      "role": "user",
      "content": "I need to solve the equation `3x + 11 = 14`. Can you help me?",
      "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:

node.js Copy

```
import fs from "fs";
import OpenAI from "openai";

const openai = new OpenAI();

async function main() {
  const response = await openai.files.content("file-abc123");

  // Extract the binary data from the Response object
  const image_data = await response.arrayBuffer();

  // Convert the binary data to a Buffer
  const image_data_buffer = Buffer.from(image_data);

  // Save the image to a specific location
  fs.writeFileSync("./my-image.png", image_data_buffer);
}

main();
```

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 and saved to a ne
        "annotations": [
          {
            "type": "file_path",
            "text": "sandbox:/mnt/data/shuffled_file.csv",
            "start_index": 167,
            "end_index": 202,
            "file_path": {
              "file_id": "file-abc123"
            }
          }
```

```
22            ...
```

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:

node.js Copy

```
const runSteps = await openai.beta.threads.runs.steps.list(
  thread.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\nresult",
              "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:

node.js Copy

```
const assistant = await openai.beta.assistants.create({
  instructions: "You are a customer support chatbot. Use your knowledge base to best respond
  model: "gpt-4-1106-preview",
  tools: [{"type": "retrieval"}]
});
```

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 at the Thread-level

node.js Copy

```
// Upload a file with an "assistants" purpose
```

```
const file = await openai.files.create({
  file: fs.createReadStream("knowledge.pdf"),
  purpose: "assistants",
});

// Add the file to the assistant
const assistant = await openai.beta.assistants.create({
  instructions: "You are a customer support chatbot. Use your knowledge base to best respon
  model: "gpt-4-1106-preview",
  tools: [{"type": "retrieval"}],
  file_ids: [file.id]
});
```

Files can also be added to a Message in a Thread. These files are only accessible within this specific thread. After having uploaded a file, you can pass the ID of this File when creating the Message:

node.js Copy

```
const message = await openai.beta.threads.messages.create(
  thread.id,
  {
    role: "user",
    content: "I can not find in the PDF manual how to turn off this device.",
    file_ids: [file.id]
  }
);
```

Maximum file size is 512MB. 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.

Deleting files

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

node.js Copy

```
const fileDeletionStatus = await openai.beta.assistants.files.del(
  assistant.id,
  file.id
);
```

Detaching the file from the assistant removes the file from the retrieval index as well.

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.

```
{
    "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 and saved to a
          "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:

node.js Copy

```
const assistant = await openai.beta.assistants.create({
  instructions: "You are a weather bot. Use the provided functions to answer questions.",
  model: "gpt-4-1106-preview",
  tools: [{
    "type": "function",
    "function": {
      "name": "getCurrentWeather",
      "description": "Get the weather in location",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {"type": "string", "description": "The city and state e.g. San Francis
          "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 state e.g. San Francis
        },
        "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\"}"
          }
        }
      ]
    }
  },
```

```
30 ...
```

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.

node.js Copy

```
const run = await openai.beta.threads.runs.submitToolOutputs(
  thread.id,
  run.id,
  {
    tool_outputs: [
      {
        tool_call_id: callIds[0],
        output: "22C",
      },
      {
        tool_call_id: callIds[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	RETRIEVAL
.c	text/x-c	✓	✓
.cpp	text/x-c++	✓	✓
.csv	application/csv	✓	
.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	✓	
.gif	image/gif	✓	
.png	image/png	✓	
.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?