

# File Search Beta


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.

## Quickstart

In this example, we'll create an assistant that can help answer questions about companies' financial statements.

### Step 1: Create a new Assistant with File Search Enabled

Create a new assistant with `file_search` enabled in the `tools` parameter of the Assistant.

```
python ▾   
1 from openai import OpenAI  
2  
3 client = OpenAI()  
4  
5 assistant = client.beta.assistants.create(  
6     name="Financial Analyst Assistant",  
7     instructions="You are an expert financial analyst. Use your knowledge base to answer q  
8     model="gpt-4-turbo",  
9     tools=[{"type": "file_search"}],  
10 )
```

Once the `file_search` tool is enabled, the model decides when to retrieve content based on user messages.

### Step 2: Upload files and add them to a Vector Store

To access your files, the `file_search` tool uses the Vector Store object. Upload your files and create a Vector Store to contain them. Once the Vector Store is created, you should poll its status until all files are out of the `in_progress` state to ensure that all content has finished processing. The SDK provides helpers to uploading and polling in one shot.

python ▾



```
1 # Create a vector store called "Financial Statements"
2 vector_store = client.beta.vector_stores.create(name="Financial Statements")
3
4 # Ready the files for upload to OpenAI
5 file_paths = ["edgar/goog-10k.pdf", "edgar/brka-10k.txt"]
6 file_streams = [open(path, "rb") for path in file_paths]
7
8 # Use the upload and poll SDK helper to upload the files, add them to the vector store,
9 # and poll the status of the file batch for completion.
10 file_batch = client.beta.vector_stores.file_batches.upload_and_poll(
11     vector_store_id=vector_store.id, files=file_streams
12 )
13
14 # You can print the status and the file counts of the batch to see the result of this o
15 print(file_batch.status)
16 print(file_batch.file_counts)
```

### Step 3: Update the assistant to use the new Vector Store

To make the files accessible to your assistant, update the assistant's `tool_resources` with the new `vector_store` id.

python ▾



```
1 assistant = client.beta.assistants.update(
2     assistant_id=assistant.id,
3     tool_resources={"file_search": {"vector_store_ids": [vector_store.id]}},
4 )
```

### Step 4: Create a thread

You can also attach files as Message attachments on your thread. Doing so will create another `vector_store` associated with the thread, or, if there is already a vector store attached to this thread, attach the new files to the existing thread vector store. When you create a Run on this thread, the file search tool will query both the `vector_store` from your assistant and the `vector_store` on the thread.

In this example, the user attached a copy of Apple's latest 10-K filing.

python ▾



```
1 # Upload the user provided file to OpenAI
2 message_file = client.files.create(
3     file=open("edgar/aapl-10k.pdf", "rb"), purpose="assistants"
4 )
5
6 # Create a thread and attach the file to the message
7 thread = client.beta.threads.create(
8     messages=[
9         {
10             "role": "user",
11             "content": "How many shares of AAPL were outstanding at the end of of October 202",
12             # Attach the new file to the message.
13             "attachments": [
14                 { "file_id": message_file.id, "tools": [{"type": "file_search"}] }
15             ],
16         }
17     ]
18 )
19
20 # The thread now has a vector store with that file in its tool resources.
21 print(thread.tool_resources.file_search)
```

Vector stores created using message attachments have a default expiration policy of 7 days after they were last active (defined as the last time the vector store was part of a run). This default exists to help you manage your vector storage costs. You can override these expiration policies at any time. Learn more [here](#).

## Step 5: Create a run and check the output

Now, create a Run and observe that the model uses the File Search tool to provide a response to the user's question.

With streaming

Without streaming

python ▾



```
1 from typing_extensions import override
2 from openai import AssistantEventHandler, OpenAI
3
4 client = OpenAI()
```

```

5
6 class EventHandler(AssistantEventHandler):
7     @override
8     def on_text_created(self, text) -> None:
9         print(f"\nassistant > ", end="", flush=True)
10
11     @override
12     def on_tool_call_created(self, tool_call):
13         print(f"\nassistant > {tool_call.type}\n", flush=True)
14
15     @override
16     def on_message_done(self, message) -> None:
17         # print a citation to the file searched
18         message_content = message.content[0].text
19         annotations = message_content.annotations
20         citations = []
21         for index, annotation in enumerate(annotations):
22             message_content.value = message_content.value.replace(
23                 annotation.text, f"[{index}]"
24             )
25             if file_citation := getattr(annotation, "file_citation", None):
26                 cited_file = client.files.retrieve(file_citation.file_id)
27                 citations.append(f"[{index}] {cited_file.filename}")
28
29         print(message_content.value)
30         print("\n".join(citations))
31
32
33 # Then, we use the stream SDK helper
34 # with the EventHandler class to create the Run
35 # and stream the response.
36
37 with client.beta.threads.runs.stream(
38     thread_id=thread.id,
39     assistant_id=assistant.id,
40     instructions="Please address the user as Jane Doe. The user has a premium account.",
41     event_handler=EventHandler(),
42 ) as stream:
43     stream.until_done()

```

Your new assistant will query both attached vector stores (one containing `goog-10k.pdf` and `brka-10k.txt`, and the other containing `aapl-10k.pdf`) and return this result from `aapl-10k.pdf`.

## How it works

The `file_search` tool implements several retrieval best practices out of the box to help you extract the right data from your files and augment the model's responses. The `file_search` tool:

- Rewrites user queries to optimize them for search.
- Breaks down complex user queries into multiple searches it can run in parallel.
- Runs both keyword and semantic searches across both assistant and thread vector stores.
- Reranks search results to pick the most relevant ones before generating the final response.

By default, the `file_search` tool uses the following settings:

- Chunk size: 800 tokens
- Chunk overlap: 400 tokens
- Embedding model: `text-embedding-3-large` at 256 dimensions
- Maximum number of chunks added to context: 20 (could be fewer)

## Known Limitations

We have a few known limitations we're working on adding support for in the coming months:

- 1 Support for modifying chunking, embedding, and other retrieval configurations.
- 2 Support for deterministic pre-search filtering using custom metadata.
- 3 Support for parsing images within documents (including images of charts, graphs, tables etc.)
- 4 Support for retrievals over structured file formats (like `csv` or `jsonl`).
- 5 Better support for summarization — the tool today is optimized for search queries.

## Vector stores

Vector Store objects give the File Search tool the ability to search your files. Adding a file to a `vector_store` automatically parses, chunks, embeds and stores the file in a vector database that's capable of both keyword and semantic search. Each `vector_store` can hold up to 10,000 files. Vector stores can be attached to both Assistants and Threads. Today, you can attach at most one vector store to an assistant and at most one vector store to a thread.

## Creating vector stores and adding files

You can create a vector store and add files to it in a single API call:


python ▾




```
1 vector_store = client.beta.vector_stores.create(  
2     name="Product Documentation",  
3     file_ids=['file_1', 'file_2', 'file_3', 'file_4', 'file_5']  
4 )
```

Adding files to vector stores is an async operation. To ensure the operation is complete, we recommend that you use the 'create and poll' helpers in our official SDKs. If you're not using the SDKs, you can retrieve the `vector_store` object and monitor its `file_counts` property to see the result of the file ingestion operation.

Files can also be added to a vector store after it's created by [creating vector store files](#).

```
python ▾   
  
1 file = client.beta.vector_stores.files.create_and_poll(  
2     vector_store_id="vs_abc123",  
3     file_id="file-abc123"  
4 )
```

Alternatively, you can add several files to a vector store by [creating batches](#) of up to 500 files.

```
python ▾   
  
1 batch = client.beta.vector_stores.file_batches.create_and_poll(  
2     vector_store_id="vs_abc123",  
3     file_ids=['file_1', 'file_2', 'file_3', 'file_4', 'file_5']  
4 )
```

Similarly, these files can be removed from a vector store by either:


- Deleting the [vector store file object](#) or,
- By deleting the underlying [file object](#) (which removes the file it from all `vector_store` and `code_interpreter` configurations across all assistants and threads in your organization)

The maximum file size is 512 MB. Each file should contain no more than 5,000,000 tokens per file (computed automatically when you attach a file).

File Search supports a variety of file formats including `.pdf`, `.md`, and `.docx`. More details on the file extensions (and their corresponding MIME-types) supported can be found in the [Supported files](#) section below.

## Attaching vector stores

You can attach vector stores to your Assistant or Thread using the `tool_resources` parameter.

```
python ▾   
1 assistant = client.beta.assistants.create(  
2     instructions="You are a helpful product support assistant and you answer questions ba  
3     model="gpt-4-turbo",  
4     tools=[{"type": "file_search"}],  
5     tool_resources={  
6         "file_search": {  
7             "vector_store_ids": ["vs_1"]  
8         }  
9     }  
10 )  
11  
12 thread = client.beta.threads.create(  
13     messages=[ { "role": "user", "content": "How do I cancel my subscription?" } ],  
14     tool_resources={  
15         "file_search": {  
16             "vector_store_ids": ["vs_2"]  
17         }  
18     }  
19 )
```

You can also attach a vector store to Threads or Assistants after they're created by updating them with the right `tool_resources`.

## Ensuring vector store readiness before creating runs

We highly recommend that you ensure all files in a `vector_store` are fully processed before you create a run. This will ensure that all the data in your `vector_store` is searchable. You can check for `vector_store` readiness by using the polling helpers in our SDKs, or by manually polling the `vector_store` object to ensure the `status` is `completed`.


As a fallback, we've built a **60 second maximum wait** in the Run object when the **thread's** vector store contains files that are still being processed. This is to ensure that any files your users upload in a thread are fully searchable before the run proceeds. This fallback wait *does not* apply to the assistant's vector store.

## Managing costs with expiration policies

The `file_search` tool uses the `vector_stores` object as its resource and you will be billed based on the [size](#) of the `vector_store` objects created. The size of the vector store object is the sum of all the parsed chunks from your files and their corresponding embeddings.

You first GB is free and beyond that, usage is billed at \$0.10/GB/day of vector storage. There are no other costs associated with vector store operations.


In order to help you manage the costs associated with these `vector_store` objects, we have added support for expiration policies in the `vector_store` object. You can set these policies when creating or updating the `vector_store` object.

```
python 
1 vector_store = client.beta.vector_stores.create_and_poll(
2     name="Product Documentation",
3     file_ids=['file_1', 'file_2', 'file_3', 'file_4', 'file_5'],
4     expires_after={
5         "anchor": "last_active_at",
6         "days": 7
7     }
8 )
```

### Thread vector stores have default expiration policies

Vector stores created using thread helpers (like `tool_resources.file_search.vector_stores` in Threads or `message.attachments` in Messages) have a default expiration policy of 7 days after they were last active (defined as the last time the vector store was part of a run).

When a vector store expires, runs on that thread will fail. To fix this, you can simply recreate a new `vector_store` with the same files and reattach it to the thread.

```
python 
1 all_files = list(client.beta.vector_stores.files.list("vs_expired"))
2
3 vector_store = client.beta.vector_stores.create(name="rag-store")
4 client.beta.threads.update(
5     "thread_abc123",
6     tool_resources={"file_search": {"vector_store_ids": [vector_store.id]}},
7 )
8
9 for file_batch in chunked(all_files, 100):
10     client.beta.vector_stores.file_batches.create_and_poll(
```



```
11     vector_store_id=vector_store.id, file_ids=[file.id for file in file_batch]
12 )
```

## Supported files

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

FILE FORMAT	MIME TYPE
.c	text/x-c
.cs	text/x-csharp
.cpp	text/x-c++
.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
.js	text/javascript

FILE FORMAT	MIME TYPE
.sh	application/x-sh
.ts	application/typescript