Package: openaiRtools (via r-universe)

June 5, 2026

Title R Client for the 'OpenAI' API

Version 0.2.2

Description Complete R implementation of OpenAI Python SDK. Provides full compatibility with OpenAI API including chat completions, embeddings, images, audio, fine-tuning, and model management.

License MIT + file LICENSE

URL https://github.com/xiaoluolorn/openaiR

BugReports https://github.com/xiaoluolorn/openaiR/issues

Depends R (>= 4.0.0)

Imports R6, httr2, jsonlite, rlang, tools, grDevices

Suggests testthat (>= 3.0.0), knitr, rmarkdown, ggplot2

VignetteBuilder knitr

Config/testthat/edition 3

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Config/pak/sysreqs libssl-dev

Repository https://xiaoluolorn.r-universe.dev

Date/Publication 2026-03-02 10:09:45 UTC

RemoteUrl https://github.com/xiaoluolorn/openairtools

RemoteRef HEAD

RemoteSha a1dff1dfc98b196d3de2ea7066935194ad1b68d8

Contents

add_upload_part *Upload a File Part (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and uploads one part of a multipart upload. Call this repeatedly for each chunk of your file.

Usage

```
add_upload_part(upload_id, data)
```

Arguments

`upload_id`	Character. **Required.** The upload session ID returned by `create_upload()`.
`data`	Raw. **Required.** The binary chunk to upload. Read from file with `readBin(con, "raw", n = chunk_size)`.

Value

An upload part object with `$id` — save all part IDs to use in `complete_upload()`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

chunk <- readBin("large_file.jsonl", "raw", n = 64 * 1024 * 1024)
part <- add_upload_part(upload_id = upload$id, data = chunk)
cat("Part ID:", part$id)

## End(Not run)
```

AssistantsClient *Assistants Client (Beta)*

Description

An Assistant is a configured AI entity with a model, instructions, and optional tools. It operates on ThreadsClient (conversations) by creating Runs that execute the assistant's logic.

Details

Client for the OpenAI Assistants API v2 (Beta). Assistants are AI agents that can use tools (code interpreter, file search, custom functions) and maintain persistent state. Access via `client$assistants`.

Typical workflow

1. Create an assistant: `client$assistants$create(model, instructions, tools)`
2. Create a thread: `client$threads$create()`
3. Add a user message: `client$threads$messages$create(thread_id, "user", "...")`
4. Create a run: `client$threads$runs$create(thread_id, assistant_id)`
5. Poll the run until status is `"completed"`
6. Read the assistant's reply: `client$threads$messages$list(thread_id)`

Methods

Public methods:

Method `new()`:

Usage:

```
AssistantsClient$new(parent)
```

Method `create()`:

Usage:

```
AssistantsClient$create(
  model,
  name = NULL,
  description = NULL,
  instructions = NULL,
  tools = NULL,
  tool_resources = NULL,
  metadata = NULL,
  temperature = NULL,
  top_p = NULL,
  response_format = NULL
)
```

Method `list()`:

Usage:

```
AssistantsClient$list(limit = NULL, order = NULL, after = NULL, before = NULL)
```

Method `retrieve()`:

Usage:

```
AssistantsClient$retrieve(assistant_id)
```

Method `update()`**:**

Usage:

```
AssistantsClient$update(assistant_id, ...)
```

Method `delete()`**:**

Usage:

```
AssistantsClient$delete(assistant_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AssistantsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`AudioClient` *Audio Client*

Description

Top-level client for the OpenAI Audio API, providing access to transcription, translation, and text-to-speech services. Access via `client$audio`.

Sub-clients

`$transcriptions` AudioTranscriptionsClient — Whisper speech-to-text

`$translations` AudioTranslationsClient — Whisper audio translation to English

`$speech` SpeechClient — Text-to-speech (TTS)

Methods

Public methods:

- `AudioClient$new()`
- `AudioClient$clone()`

Method `new()`**:**

Usage:

```
AudioClient$new(parent)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AudioClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

AudioTranscriptionsClient
Audio Transcriptions Client

Description

Transcribes audio files to text using OpenAI's Whisper model. Access via `client$audio$transcriptions`.

Methods

Public methods:

- `AudioTranscriptionsClient$new()`
- `AudioTranscriptionsClient$create()`
- `AudioTranscriptionsClient$clone()`

Method `new()`:

Usage:

```
AudioTranscriptionsClient$new(parent)
```

Method `create()`:

Usage:

```
AudioTranscriptionsClient$create(
  file,
  model = "whisper-1",
  language = NULL,
  prompt = NULL,
  response_format = NULL,
  temperature = NULL,
  timestamp_granularities = NULL
)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
AudioTranscriptionsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

AudioTranslationsClient

Audio Translations Client

Description

Translates audio from any supported language into English text using Whisper. Access via `client$audio$translations`.

Methods

Public methods:

- `AudioTranslationsClient$new()`
- `AudioTranslationsClient$create()`
- `AudioTranslationsClient$clone()`

Method `new()`:

Usage:

```
AudioTranslationsClient$new(parent)
```

Method `create()`:

Usage:

```
AudioTranslationsClient$create(
  file,
  model = "whisper-1",
  prompt = NULL,
  response_format = NULL,
  temperature = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
AudioTranslationsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`BatchClient` *Batch Client*

Description

The Batch API accepts a JSONL file of requests, processes them within a 24-hour window, and returns the results in an output file. Suitable for: bulk embeddings, offline evaluation, large-scale text processing, or any task that does not require immediate results.

Details

Client for the OpenAI Batch API. Process large volumes of API requests asynchronously at 50% lower cost than synchronous calls. Access via `client$batch`.

Workflow

1. Create a JSONL file where each line is one API request (see format below)
2. Upload the file: `client$files$create(file, purpose = "batch")`
3. Create a batch: `client$batch$create(input_file_id, endpoint)`
4. Poll status: `client$batch$retrieve(batch_id)`
5. Download results: `client$files$content(batch$output_file_id)`

JSONL request format

Each line in the input file must be:

```
{"custom_id": "req-1", "method": "POST",
 "url": "/v1/chat/completions",
 "body": {"model": "gpt-4o-mini",
          "messages": [{"role": "user", "content": "Hello!"}]}}
```

Methods

Public methods:

- `BatchClient$new()`
- `BatchClient$create()`
- `BatchClient$list()`
- `BatchClient$retrieve()`
- `BatchClient$cancel()`
- `BatchClient$clone()`

Method `new()`:

Usage:

```
BatchClient$new(parent)
```

Method `create()`**:**

Usage:

```
BatchClient$create(
  input_file_id,
  endpoint,
  completion_window = "24h",
  metadata = NULL
)
```

Method `list()`**:**

Usage:

```
BatchClient$list(after = NULL, limit = NULL)
```

Method `retrieve()`**:**

Usage:

```
BatchClient$retrieve(batch_id)
```

Method `cancel()`**:**

Usage:

```
BatchClient$cancel(batch_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
BatchClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`cancel_batch` *Cancel a Batch Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and cancels a batch job.

Usage

```
cancel_batch(batch_id)
```

Arguments

`batch_id`	Character. **Required.** The batch ID to cancel.

Value

A batch object with `$status = "cancelling"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- cancel_batch("batch_abc123")
cat("Status:", result$status)

## End(Not run)
```

`cancel_fine_tuning_job` *Cancel a Fine-tuning Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and cancels a running or queued fine-tuning job.

Usage

```
cancel_fine_tuning_job(fine_tuning_job_id)
```

Arguments

`fine_tuning_job_id`
: Character. **Required.** The job ID to cancel.

Value

The fine-tuning job object with `$status = "cancelled"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- cancel_fine_tuning_job("ftjob-abc123")
cat("Status:", result$status) # "cancelled"

## End(Not run)
```

cancel_response *Cancel a Streaming Response (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and cancels an in-progress response.

Usage

```
cancel_response(response_id)
```

Arguments

`response_id`	Character. **Required.** The response ID to cancel.

Value

The cancelled response object.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
cancel_response("resp_abc123")

## End(Not run)
```

cancel_run *Cancel a Run (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and cancels a running or queued run.

Usage

```
cancel_run(thread_id, run_id)
```

Arguments

`thread_id`	Character. **Required.** The thread ID.
`run_id`	Character. **Required.** The run ID to cancel.

Value

A run object with `$status = "cancelling"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
cancel_run("thread_abc123", "run_abc123")

## End(Not run)
```

`cancel_upload` *Cancel a Multipart Upload (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and cancels an in-progress multipart upload.

Usage

```
cancel_upload(upload_id)
```

Arguments

`upload_id`	Character. **Required.** The upload session ID to cancel.

Value

An upload object with `$status = "cancelled"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- cancel_upload("upload_abc123")
cat("Status:", result$status) # "cancelled"

## End(Not run)
```

ChatClient *Chat Completions Client*

Description

Client for OpenAI Chat Completions API. Access via `client$chat`.

Methods

Public methods:

- `ChatClient$new()`
- `ChatClient$clone()`

Method `new()`:

Usage:

```
ChatClient$new(parent)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ChatClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ChatCompletionsClient *Chat Completions Interface*

Description

Provides methods to create, manage and retrieve chat completions. Access via `client$chat$completions`.

Methods

Public methods:

- `ChatCompletionsClient$new()`
- `ChatCompletionsClient$create()`
- `ChatCompletionsClient$retrieve()`
- `ChatCompletionsClient$update()`
- `ChatCompletionsClient$list()`
- `ChatCompletionsClient$delete()`
- `ChatCompletionsClient$clone()`

Method `new()`:

Usage:

```
ChatCompletionsClient$new(parent)
```

Method `create()`**:**

Usage:

```
ChatCompletionsClient$create(
  messages,
  model = "gpt-3.5-turbo",
  frequency_penalty = NULL,
  logit_bias = NULL,
  logprobs = NULL,
  top_logprobs = NULL,
  max_tokens = NULL,
  max_completion_tokens = NULL,
  n = NULL,
  presence_penalty = NULL,
  response_format = NULL,
  seed = NULL,
  stop = NULL,
  stream = NULL,
  stream_options = NULL,
  temperature = NULL,
  top_p = NULL,
  tools = NULL,
  tool_choice = NULL,
  parallel_tool_calls = NULL,
  user = NULL,
  store = NULL,
  metadata = NULL,
  callback = NULL,
  ...
)
```

Method `retrieve()`**:**

Usage:

```
ChatCompletionsClient$retrieve(completion_id)
```

Method `update()`**:**

Usage:

```
ChatCompletionsClient$update(completion_id, metadata = NULL)
```

Method `list()`**:**

Usage:

```
ChatCompletionsClient$list(
  model = NULL,
  after = NULL,
  limit = NULL,
```

```
  order = NULL,
  metadata = NULL
)
```

Method `delete()`**:**

Usage:

```
ChatCompletionsClient$delete(completion_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ChatCompletionsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ChatCompletionsMessagesClient`

Chat Completions Messages Client

Description

Lists messages from stored chat completions. Access via `client$chat$completions$messages`.

Methods

Public methods:

- `ChatCompletionsMessagesClient$new()`
- `ChatCompletionsMessagesClient$list()`
- `ChatCompletionsMessagesClient$clone()`

Method `new()`**:**

Usage:

```
ChatCompletionsMessagesClient$new(parent)
```

Method `list()`**:**

Usage:

```
ChatCompletionsMessagesClient$list(
  completion_id,
  after = NULL,
  limit = NULL,
  order = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ChatCompletionsMessagesClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`complete_upload` *Complete a Multipart Upload (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and finalizes a multipart upload. Assembles all uploaded parts into a single file.

Usage

```
complete_upload(upload_id, part_ids, ...)
```

Arguments

`upload_id`	Character. **Required.** The upload session ID.
`part_ids`	List of character strings. **Required.** Part IDs collected from `add_upload_part()` calls, in the correct file order.
`...`	Additional parameters such as `md5` (integrity checksum).

Value

A standard File object with `$id` that can be used in fine-tuning, assistants, or batch API calls, just like a file from `upload_file()`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# After uploading all parts, complete the upload:
file_obj <- complete_upload(
  upload_id = upload$id,
  part_ids  = list("part-001", "part-002", "part-003")
)
cat("File ID:", file_obj$id)

## End(Not run)
```

`CompletionsClient` *Completions Client (Legacy)*

Description

Note: This is the legacy "instruct" style API. For most use cases, use ChatCompletionsClient (`client$chat$completions`) instead. The Completions API is suitable only for `"gpt-3.5-turbo-instruct"` and `"davinci-002"`. It takes a raw text prompt and returns a completion.

Details

Client for the OpenAI Completions API (legacy text completion endpoint). Access via `client$completions`.

Methods

Public methods:

- `CompletionsClient$new()`
- `CompletionsClient$create()`
- `CompletionsClient$clone()`

Method `new()`:

Usage:

```
CompletionsClient$new(parent)
```

Method `create()`:

Usage:

```
CompletionsClient$create(
  prompt,
  model = "gpt-3.5-turbo-instruct",
  max_tokens = NULL,
  temperature = NULL,
  top_p = NULL,
  n = NULL,
  stream = NULL,
  logprobs = NULL,
  echo = NULL,
  stop = NULL,
  presence_penalty = NULL,
  frequency_penalty = NULL,
  best_of = NULL,
  logit_bias = NULL,
  user = NULL,
  callback = NULL
)
```

Method `clone()`:

The objects of this class are cloneable with this method.

Usage:

```
CompletionsClient$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`create_assistant` *Create an Assistant (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$assistants$create()`.

Usage

```
create_assistant(model, ...)
```

Arguments

`model`	Character. **Required.** Model ID to power the assistant (e.g. `"gpt-4o"`, `"gpt-4o-mini"`).
`...`	Additional parameters passed to AssistantsClient`$create()`: `name`, `description`, `instructions`, `tools`, `tool_resources`, `metadata`, `temperature`, `top_p`, `response_format`.

Value

An assistant object with `$id` (save this for future use), `$name`, `$model`, `$instructions`, and `$tools`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

asst <- create_assistant(
  model        = "gpt-4o",
  name         = "Stats Helper",
  instructions = "You help with statistical analysis in R and Python.",
  tools        = list(list(type = "code_interpreter"))
)
cat("Created assistant ID:", asst$id)

## End(Not run)
```

`create_batch` *Create a Batch Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$batch$create()`.

Usage

```
create_batch(input_file_id, endpoint, ...)
```

Arguments

`input_file_id`	Character. **Required.** File ID of the uploaded JSONL batch input file (uploaded with `purpose = "batch"`).
`endpoint`	Character. **Required.** API endpoint for each request. One of `"/v1/chat/completions"`, `"/v1/embeddings"`, `"/v1/completions"`.
`...`	Additional parameters passed to BatchClient`$create()`, such as `completion_window` (default `"24h"`) and `metadata`.

Value

A batch object with `$id` and `$status`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

batch <- create_batch(
  input_file_id = "file-abc123",
  endpoint      = "/v1/chat/completions"
)
cat("Batch ID:", batch$id)

## End(Not run)
```

`create_chat_completion` *Create a Chat Completion (Convenience Function)*

Description

A shortcut that automatically creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$chat$completions$create()`.

Usage

```
create_chat_completion(messages, model = "gpt-3.5-turbo", ...)
```

Arguments

`messages`	**Required.** List of message objects. Each must have `role` (`"system"`, `"user"`, `"assistant"`) and `content` fields.
`model`	Character. Model ID. Default: `"gpt-3.5-turbo"`.
`...`	Additional parameters passed to ChatCompletionsClient`$create()`, such as `temperature`, `max_tokens`, `stream`, etc.

Value

A chat completion list object. Key fields:

- `$choices[[1]]$message$content` — The generated text
- `$usage$total_tokens` — Total tokens consumed

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

response <- create_chat_completion(
  messages = list(list(role = "user", content = "What is machine learning?")),
  model    = "gpt-4o"
)
cat(response$choices[[1]]$message$content)

# With extra parameters
response <- create_chat_completion(
  messages    = list(list(role = "user", content = "Write a poem")),
  model       = "gpt-4o",
  temperature = 1.2,
  max_tokens  = 200
)

## End(Not run)
```

`create_completion` *Create a Legacy Text Completion (Convenience Function)*

Description

Note: This is the legacy Completions API. For most tasks, use `create_chat_completion()` instead, which supports system prompts, conversation history, and more capable models.

Usage

```
create_completion(prompt, model = "gpt-3.5-turbo-instruct", ...)
```

Arguments

`prompt`	Character. **Required.** The text prompt to complete.
`model`	Character. Model ID. Only `"gpt-3.5-turbo-instruct"`, `"davinci-002"`, and `"babbage-002"` support this endpoint. Default: `"gpt-3.5-turbo-instruct"`.
`...`	Additional parameters passed to CompletionsClient`$create()`, such as `max_tokens`, `temperature`, `n`, `stop`, `stream`, etc.

Details

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$completions$create()`.

Value

A named list. Access the generated text via `$choices[[1]]$text`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Complete a sentence
resp <- create_completion(
  prompt     = "The most important assumption of OLS regression is",
  model      = "gpt-3.5-turbo-instruct",
  max_tokens = 100
)
cat(resp$choices[[1]]$text)

## End(Not run)
```

create_embedding *Create Text Embeddings (Convenience Function)*

Description

Shortcut that automatically creates an `OpenAI` client and calls `client$embeddings$create()`. The API key is read from the `OPENAI_API_KEY` environment variable.

Usage

```
create_embedding(input, model = "text-embedding-ada-002", ...)
```

Arguments

`input`	**Required.** A character string or list of strings to embed.
`model`	Character. Embedding model. Default: `"text-embedding-ada-002"`.
`...`	Additional parameters passed to `EmbeddingsClient$create()`, such as `dimensions` or `encoding_format`.

Value

A named list with `$data[[i]]$embedding` containing the numeric embedding vectors.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Single embedding
resp <- create_embedding("Hello world", model = "text-embedding-3-small")
vec <- unlist(resp$data[[1]]$embedding)
cat("Vector length:", length(vec))

## End(Not run)
```

create_fine_tuning_job *Create a Fine-tuning Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$fine_tuning$jobs$create()`.

Usage

```
create_fine_tuning_job(training_file, model = "gpt-3.5-turbo", ...)
```

Arguments

`training_file`	Character. **Required.** File ID of the uploaded training JSONL file (e.g. from `upload_file(..., purpose = "fine-tune")`).
`model`	Character. Base model to fine-tune: `"gpt-3.5-turbo"`, `"gpt-4o-mini"`, `"gpt-4o"`, etc. Default: `"gpt-3.5-turbo"`.
`...`	Additional parameters passed to FineTuningJobsClient`$create()`, such as `suffix`, `hyperparameters` (list with `n_epochs`, `batch_size`), `validation_file`, `seed`.

Value

A fine-tuning job object with `$id` and `$status`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

job <- create_fine_tuning_job(
  training_file   = "file-abc123",
  model           = "gpt-3.5-turbo",
  suffix          = "my-assistant",
  hyperparameters = list(n_epochs = 3)
)
cat("Job ID:", job$id)

## End(Not run)
```

`create_image` *Generate an Image from Text (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$images$create()`.

Usage

```
create_image(prompt, model = "dall-e-3", ...)
```

Arguments

`prompt`	Character. **Required.** Text description of the desired image. For DALL-E 3, up to 4000 characters.
`model`	Character. Image model: `"dall-e-3"` (default) or `"dall-e-2"`.
`...`	Additional parameters passed to ImagesClient`$create()`, such as `n`, `size` (`"1024x1024"`, `"1792x1024"`, `"1024x1792"`), `quality` (`"standard"` or `"hd"`), `style` (`"vivid"` or `"natural"`), `response_format` (`"url"` or `"b64_json"`).

Value

A list with `$data[[1]]$url` containing the image URL, and `$data[[1]]$revised_prompt` with the prompt used by DALL-E 3.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Generate a standard image
resp <- create_image("A futuristic chart showing economic data, neon style")
cat(resp$data[[1]]$url)

# HD landscape
resp <- create_image(
  prompt  = "A detailed map of global trade routes",
  model   = "dall-e-3",
  size    = "1792x1024",
  quality = "hd",
  style   = "natural"
)
cat(resp$data[[1]]$url)

## End(Not run)
```

---

create_image_edit *Edit an Image with DALL-E (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$images$edit()`. Edits a local PNG image based on a text prompt and optional mask.

Usage

```
create_image_edit(image, prompt, mask = NULL, ...)
```

Arguments

`image`	Character. **Required.** Path to the local square PNG to edit (less than 4 MB).
`prompt`	Character. **Required.** Description of the desired edit.
`mask`	Character or NULL. Path to a mask PNG. Transparent pixels indicate which region to edit. Default: NULL (entire image).
`...`	Additional parameters passed to ImagesClient`$edit()`, such as `n` (number of images), `size`, `response_format`, `user`.

Value

A list with `$data` containing the edited image object(s). Access the URL via `$data[[1]]$url`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

resp <- create_image_edit(
  image  = "photo.png",
  prompt = "Add a mountain range in the background",
  mask   = "sky_mask.png"
)
cat(resp$data[[1]]$url)

## End(Not run)
```

---

create_image_variation

Create Image Variations with DALL-E (Convenience Function)

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$images$create_variation()`. Generates one or more variations of an existing image.

Usage

```
create_image_variation(image, model = "dall-e-2", ...)
```

Arguments

`image`	Character. **Required.** Path to the local square PNG image (less than 4 MB).
`model`	Character. Currently only `"dall-e-2"` is supported. Default: `"dall-e-2"`.
`...`	Additional parameters passed to ImagesClient`$create_variation()`, such as `n` (number of variations, 1–10), `size`, `response_format`.

Value

A list with `$data` containing the generated variation image(s). Access the URL via `$data[[1]]$url`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Generate 2 variations of your logo
resp <- create_image_variation(
  image = "logo.png",
  n     = 2,
  size  = "512x512"
)
cat(resp$data[[1]]$url)
cat(resp$data[[2]]$url)

## End(Not run)
```

`create_message` *Add a Message to a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and adds a message to a thread.

Usage

```
create_message(thread_id, role, content, ...)
```

Arguments

`thread_id`	Character. **Required.** The thread ID.
`role`	Character. **Required.** `"user"` or `"assistant"`.
`content`	Character or list. **Required.** The message text, or a list of content parts for multimodal messages.
`...`	Additional parameters passed to MessagesClient`$create()`, such as `attachments` and `metadata`.

Value

A message object with `$id` and `$content`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

msg <- create_message(
  thread_id = "thread_abc123",
  role      = "user",
```

```
  content   = "What is the difference between FE and RE estimators?"
)
cat("Message ID:", msg$id)

## End(Not run)
```

`create_moderation` *Check Content for Policy Violations (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$moderations$create()`.

Usage

```
create_moderation(input, model = "omni-moderation-latest")
```

Arguments

`input`	**Required.** A character string or list of strings to moderate. Example: `"Kill all humans"` or `list("Hello", "I hate you")`.
`model`	Character. Moderation model to use: `"omni-moderation-latest"` (default, supports images) or `"text-moderation-latest"`.

Details

This API is **free** and does not consume tokens. Use it to screen user-generated content before passing it to other APIs.

Value

A list with `$results` — a list of result objects, one per input. Each result has:

- `$flagged` — Logical, `TRUE` if content violates policy
- `$categories` — Named list of boolean flags per category
- `$category_scores` — Named list of scores (0–1) per category

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Quick single-text check
result <- create_moderation("I love everyone!")
cat("Flagged:", result$results[[1]]$flagged) # FALSE

# Screen multiple texts from user input
texts <- list("normal message", "harmful content example")
```

```
result <- create_moderation(texts)
for (i in seq_along(result$results)) {
  cat("Text", i, "flagged:", result$results[[i]]$flagged, "\n")
}

## End(Not run)
```

`create_multimodal_message` *Build a Multimodal User Message (Text + Images)*

Description

Convenience function that assembles a complete `"user"` message object containing both text and one or more images. Automatically handles URL vs. local file detection.

Usage

```
create_multimodal_message(text = NULL, images = NULL, detail = "auto")
```

Arguments

`text`	Character or `NULL`. The text prompt accompanying the image(s). If `NULL`, only images are sent (less common).
`images`	List of image sources. Each element can be: • A **URL string** starting with `"http://"` or `"https://"` — passed to `image_from_url` • A **local file path string** — passed to `image_from_file` • A **pre-built content part** from `image_from_url`, `image_from_file`, or `image_from_plot` (these are passed through as-is) Default: `NULL` (text-only message).
`detail`	Character. Detail level applied to all images supplied as strings. Ignored for pre-built content parts. `"auto"` (default), `"low"`, or `"high"`.

Details

Pass the returned object (or a list of such objects) directly to `client$chat$completions$create(messages = ...)`.

Value

A named list representing a `"user"` message:

```
list(
  role    = "user",
  content = list(
    list(type = "text",      text = <text>),
```

```
    list(type = "image_url", image_url = list(url = ..., detail = ...)),
    ...
  )
)
```

See Also

image_from_url(), image_from_file(), image_from_plot()

Examples

```
## Not run:
library(openaiRtools)
client <- OpenAI$new(api_key = "sk-xxxxxx")

# --- URL image ---
msg <- create_multimodal_message(
  text   = "What is shown in this chart?",
  images = list("https://example.com/gdp_chart.png")
)
response <- client$chat$completions$create(messages = list(msg), model = "gpt-4o")

# --- Local file ---
msg <- create_multimodal_message(
  text   = "Identify any statistical issues in this residual plot.",
  images = list("output/resid_plot.png"),
  detail = "high"
)

# --- Multiple images (compare two charts) ---
msg <- create_multimodal_message(
  text   = "Compare these two regression diagnostics plots.",
  images = list("plot_model1.png", "plot_model2.png"),
  detail = "high"
)

# --- Mix of pre-built parts ---
library(ggplot2)
p <- ggplot(mtcars, aes(wt, mpg)) +
  geom_point()
msg <- create_multimodal_message(
  text   = "Describe the scatter pattern.",
  images = list(image_from_plot(p, dpi = 180))
)

## End(Not run)
```

---

create_response *Create a Response (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$responses$create()`.

Usage

```
create_response(model, input, ...)
```

Arguments

`model`	Character. **Required.** Model ID (e.g. `"gpt-4o"`, `"gpt-4o-mini"`, `"o1"`).
`input`	**Required.** A text string or list of message objects as the prompt.
`...`	Additional parameters passed to ResponsesClient`$create()`, such as `instructions`, `previous_response_id` (for multi-turn), `tools`, `temperature`, `max_output_tokens`, `stream`, `store`.

Details

The Responses API is OpenAI's newer, simpler alternative to Chat Completions. It natively supports multi-turn conversations using `previous_response_id` (no need to resend message history), and includes built-in web search, file search, and other tools.

Value

A response object. Access the text via `$output[[1]]$content[[1]]$text`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Simple text response
resp <- create_response(
  model = "gpt-4o",
  input = "What is the difference between OLS and IV?"
)
cat(resp$output[[1]]$content[[1]]$text)

# Multi-turn: continue conversation using previous response ID
resp1 <- create_response("gpt-4o", "What is GMM?")
resp2 <- create_response(
  model               = "gpt-4o",
  input               = "Give me an R code example.",
  previous_response_id = resp1$id
)
cat(resp2$output[[1]]$content[[1]]$text)

## End(Not run)
```

`create_run` *Create a Run (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and creates a run on a thread. Poll the run with `retrieve_run()` until status is `"completed"`.

Usage

```
create_run(thread_id, assistant_id, ...)
```

Arguments

`thread_id`	Character. **Required.** The thread ID.
`assistant_id`	Character. **Required.** The assistant ID to run.
`...`	Additional parameters passed to RunsClient`$create()`, such as `instructions`, `model`, `tools`, `stream`, `callback`.

Value

A run object with `$id` and `$status`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

run <- create_run(
  thread_id    = "thread_abc123",
  assistant_id = "asst_abc123"
)
cat("Run ID:", run$id, "Status:", run$status)

# Poll until done
repeat {
  run <- retrieve_run("thread_abc123", run$id)
  if (run$status %in% c("completed", "failed", "cancelled")) break
  Sys.sleep(2)
}

## End(Not run)
```

`create_speech` *Convert Text to Speech (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$audio$speech$create()`. Returns raw binary audio data that should be saved with `writeBin()`.

Usage

```
create_speech(input, model = "tts-1", voice = "alloy", ...)
```

Arguments

`input`	Character. **Required.** The text to synthesize (max 4096 chars).
`model`	Character. TTS model: `"tts-1"` (fast) or `"tts-1-hd"` (higher quality). Default: `"tts-1"`.
`voice`	Character. Voice style. One of: `"alloy"`, `"ash"`, `"coral"`, `"echo"`, `"fable"`, `"onyx"`, `"nova"`, `"sage"`, `"shimmer"`. Default: `"alloy"`.
`...`	Additional parameters passed to SpeechClient`$create()`, such as `response_format` (`"mp3"`, `"wav"`, `"flac"`, etc.) and speed (0.25–4.0).

Value

A `raw` vector of binary audio data. Save using `writeBin()`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Generate and save to MP3
audio <- create_speech(
  input = "The quick brown fox jumps over the lazy dog.",
  model = "tts-1",
  voice = "nova"
)
writeBin(audio, "output.mp3")

# High-quality WAV with slower speed
audio <- create_speech(
  input           = "Welcome to the lecture on macroeconomics.",
  model           = "tts-1-hd",
  voice           = "onyx",
  response_format = "wav",
  speed           = 0.9
)
writeBin(audio, "lecture_intro.wav")
```

```
## End(Not run)
```

`create_thread` *Create a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and creates a new thread.

Usage

```
create_thread(...)
```

Arguments

`...`	Parameters passed to ThreadsClient`$create()`: `messages` (list of initial messages), `tool_resources`, `metadata`.

Value

A thread object with `$id` (save this for subsequent calls).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

thread <- create_thread()
cat("Thread ID:", thread$id)

# With an initial message
thread <- create_thread(
  messages = list(list(role = "user", content = "Hello!"))
)

## End(Not run)
```

`create_transcription` *Transcribe Audio to Text (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$audio$transcriptions$create()`.

Usage

```
create_transcription(file, model = "whisper-1", ...)
```

Arguments

`file`	Character. **Required.** Path to the local audio file. Supported: flac, mp3, mp4, mpeg, mpga, m4a, ogg, wav, webm (max 25 MB).
`model`	Character. Whisper model. Default: `"whisper-1"`.
`...`	Additional parameters passed to AudioTranscriptionsClient`$create()`, such as `language`, `prompt`, `response_format`, `temperature`, `timestamp_granularities`.

Value

A list with `$text` containing the transcribed text (for default JSON format), or a character string for `response_format = "text"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Basic transcription
result <- create_transcription("meeting.mp3")
cat(result$text)

# With language hint and plain text output
text <- create_transcription(
  file            = "lecture.m4a",
  model           = "whisper-1",
  language        = "en",
  response_format = "text"
)
cat(text)

## End(Not run)
```

create_translation *Translate Audio to English Text (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$audio$translations$create()`. Always produces English output regardless of source language.

Usage

```
create_translation(file, model = "whisper-1", ...)
```

Arguments

`file`	Character. **Required.** Path to the local audio file. Supported: flac, mp3, mp4, mpeg, mpga, m4a, ogg, wav, webm (max 25 MB).
`model`	Character. Whisper model. Default: `"whisper-1"`.
`...`	Additional parameters passed to AudioTranslationsClient`$create()`, such as `prompt`, `response_format`, `temperature`.

Value

A list with `$text` containing the English translation (for default JSON format), or a character string for `response_format = "text"`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Translate a Chinese audio file to English
result <- create_translation("chinese_speech.mp3")
cat(result$text) # Output is always in English

## End(Not run)
```

create_upload *Create a Multipart Upload Session (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and initializes a multipart upload session. Use this when your file exceeds the 512 MB single-upload limit.

Usage

```
create_upload(purpose, filename, bytes, ...)
```

Arguments

purpose	Character. **Required.** The intended use: "assistants", "batch", or "fine-tune".
filename	Character. **Required.** The original name of the file.
bytes	Integer. **Required.** Total file size in bytes (file.info("myfile.jsonl")$size).
...	Additional parameters passed to UploadsClient$create(), such as mime_type.

Value

An upload object with $id (use in add_upload_part() and complete_upload()).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

file_size <- file.info("huge_training.jsonl")$size
upload <- create_upload(
  purpose   = "fine-tune",
  filename  = "huge_training.jsonl",
  bytes     = file_size,
  mime_type = "application/json"
)
cat("Upload ID:", upload$id)

## End(Not run)
```

---

create_vector_store *Create a Vector Store (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and creates a new vector store.

Usage

```
create_vector_store(...)
```

Arguments

...	Parameters passed to VectorStoresClient$create(): name, file_ids, expires_after, chunking_strategy, metadata.

Value

A vector store object with `$id` (save this), `$name`, `$status`, and `$file_counts`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

vs <- create_vector_store(
  name     = "Economics Literature",
  file_ids = list("file-abc123", "file-def456")
)
cat("Vector Store ID:", vs$id)

## End(Not run)
```

---

delete_assistant *Delete an Assistant (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and permanently deletes an assistant.

Usage

```
delete_assistant(assistant_id)
```

Arguments

`assistant_id`	Character. **Required.** The ID of the assistant to delete.

Value

A list with `$deleted` (`TRUE` if successful) and `$id`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- delete_assistant("asst_abc123")
if (result$deleted) cat("Assistant deleted.")

## End(Not run)
```

delete_file *Delete an Uploaded File (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and deletes a file.

Usage

```
delete_file(file_id)
```

Arguments

`file_id`	Character. **Required.** The file ID to delete.

Value

A list with `$deleted` (`TRUE` if successful) and `$id`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- delete_file("file-abc123")
if (result$deleted) cat("File deleted successfully.")

## End(Not run)
```

delete_model *Delete a Fine-Tuned Model (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and deletes a fine-tuned model you own. Only models you created via fine-tuning can be deleted.

Usage

```
delete_model(model)
```

Arguments

`model`	Character. **Required.** The fine-tuned model ID to delete. Fine-tuned model IDs have the format `"ft:gpt-3.5-turbo:org:suffix:id"`.

Value

A list with $id (the deleted model ID) and $deleted (TRUE if successful).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- delete_model("ft:gpt-3.5-turbo:myorg:experiment:abc123")
if (result$deleted) cat("Model deleted successfully.\n")

## End(Not run)
```

---

delete_response *Delete a Stored Response (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and deletes a stored response.

Usage

```
delete_response(response_id)
```

Arguments

response_id	Character. **Required.** The response ID to delete.

Value

A list with $deleted (TRUE if successful).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- delete_response("resp_abc123")
if (result$deleted) cat("Response deleted.")

## End(Not run)
```

delete_thread *Delete a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and permanently deletes a thread.

Usage

```
delete_thread(thread_id)
```

Arguments

`thread_id`	Character. **Required.** The thread ID to delete.

Value

A list with `$deleted` (`TRUE`) and `$id`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
result <- delete_thread("thread_abc123")
if (result$deleted) cat("Thread deleted.")

## End(Not run)
```

delete_vector_store *Delete a Vector Store (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and deletes a vector store. This removes the search index but does NOT delete the underlying files.

Usage

```
delete_vector_store(vector_store_id)
```

Arguments

`vector_store_id`	Character. **Required.** The vector store ID.

Value

A list with `$deleted` (`TRUE`) and `$id`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

result <- delete_vector_store("vs_abc123")
if (result$deleted) cat("Vector store deleted.")

## End(Not run)
```

EmbeddingsClient *Embeddings Client*

Description

Provides text embedding (vectorization) via the OpenAI Embeddings API. Access via `client$embeddings`.

Details

Embeddings are numerical vector representations of text that capture semantic meaning. Similar texts produce similar vectors. Common uses include semantic search, clustering, classification, and recommendation systems.

Methods

Public methods:

- `EmbeddingsClient$new()`
- `EmbeddingsClient$create()`
- `EmbeddingsClient$clone()`

Method `new()`:

Usage:

```
EmbeddingsClient$new(parent)
```

Method `create()`:

Usage:

```
EmbeddingsClient$create(
  input,
  model = "text-embedding-ada-002",
  encoding_format = NULL,
  dimensions = NULL,
  user = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
EmbeddingsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`FilesClient` *Files Client*

Description

Client for the OpenAI Files API. Upload, list, retrieve, and delete files that can be used with fine-tuning, assistants, batch processing, and more. Access via `client$files`.

Methods

`$create(file, purpose)` Upload a file

`$list(...)` List uploaded files, optionally filter by purpose

`$retrieve(file_id)` Get metadata for a specific file

`$delete(file_id)` Delete a file

`$content(file_id)` Download the raw content of a file

`$wait_for_processing(file_id, ...)` Poll until a file is processed

Methods

Public methods:

- `FilesClient$new()`
- `FilesClient$create()`
- `FilesClient$list()`
- `FilesClient$retrieve()`
- `FilesClient$delete()`
- `FilesClient$content()`
- `FilesClient$wait_for_processing()`
- `FilesClient$clone()`

Method `new()`**:**

Usage:

```
FilesClient$new(parent)
```

Method `create()`**:**

Usage:

```
FilesClient$create(file, purpose)
```

Method `list()`**:**

Usage:

```
FilesClient$list(purpose = NULL, limit = NULL, after = NULL, order = NULL)
```

Method `retrieve()`**:**

Usage:

```
FilesClient$retrieve(file_id)
```

Method `delete()`**:**

Usage:

```
FilesClient$delete(file_id)
```

Method `content()`**:**

Usage:

```
FilesClient$content(file_id)
```

Method `wait_for_processing()`**:**

Usage:

```
FilesClient$wait_for_processing(file_id, timeout = 300, poll_interval = 5)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FilesClient$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`FineTuningCheckpointsClient`

Fine-tuning Checkpoints Client

Description

Retrieves intermediate model checkpoints created during fine-tuning. A checkpoint is saved after each training epoch. Access via `client$fine_tuning$jobs$checkpoints`.

Methods

Public methods:

- `FineTuningCheckpointsClient$new()`
- `FineTuningCheckpointsClient$list()`
- `FineTuningCheckpointsClient$clone()`

Method `new()`**:**

Usage:

```
FineTuningCheckpointsClient$new(parent)
```

Method `list()`**:**

Usage:

```
FineTuningCheckpointsClient$list(
  fine_tuning_job_id,
  after = NULL,
  limit = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FineTuningCheckpointsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`FineTuningClient` *Fine-tuning Client*

Description

Top-level client for the OpenAI Fine-tuning API. Fine-tuning lets you train a custom version of a GPT model on your own examples. Access via `client$fine_tuning`.

Sub-clients

`$jobs` FineTuningJobsClient — Create and manage fine-tuning jobs

Workflow

1. Upload training data (JSONL file) via `client$files$create(purpose = "fine-tune")`
2. Create a fine-tuning job with `client$fine_tuning$jobs$create()`
3. Monitor progress with `client$fine_tuning$jobs$retrieve()` or `$list_events()`
4. Use the resulting model ID in chat completions once status is `"succeeded"`

Methods

Public methods:

- `FineTuningClient$new()`
- `FineTuningClient$clone()`

Method `new()`**:**

Usage:

```
FineTuningClient$new(parent)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FineTuningClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`FineTuningJobsClient` *Fine-tuning Jobs Client*

Description

Manage fine-tuning jobs — create, list, retrieve, cancel, and monitor events. Access via `client$fine_tuning$jobs`.

Methods

Public methods:

- `FineTuningJobsClient$new()`
- `FineTuningJobsClient$create()`
- `FineTuningJobsClient$list()`
- `FineTuningJobsClient$retrieve()`
- `FineTuningJobsClient$cancel()`
- `FineTuningJobsClient$list_events()`
- `FineTuningJobsClient$clone()`

Method `new()`**:**

Usage:

```
FineTuningJobsClient$new(parent)
```

Method `create()`**:**

Usage:

```
FineTuningJobsClient$create(
  training_file,
  model = "gpt-3.5-turbo",
  hyperparameters = NULL,
  suffix = NULL,
  validation_file = NULL,
  integrations = NULL,
  seed = NULL,
  method = NULL
)
```

Method `list()`**:**

Usage:

```
FineTuningJobsClient$list(after = NULL, limit = NULL)
```

Method `retrieve()`**:**

Usage:

```
FineTuningJobsClient$retrieve(fine_tuning_job_id)
```

Method `cancel()`**:**

Usage:

```
FineTuningJobsClient$cancel(fine_tuning_job_id)
```

Method `list_events()`**:**

Usage:

```
FineTuningJobsClient$list_events(
  fine_tuning_job_id,
  after = NULL,
  limit = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
FineTuningJobsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`image_from_file` *Create Image Content from a Local File*

Description

Reads a local image file, encodes it as Base64, and wraps it in a data URI so it can be sent directly to the model without hosting the file anywhere. This is the recommended approach for local figures, screenshots, or any image not accessible via a public URL.

Usage

```
image_from_file(file_path, mime_type = NULL, detail = "auto")
```

Arguments

`file_path`	Character. Absolute or relative path to a local image file. Supported formats: `.jpg`/`.jpeg`, `.png`, `.gif` (static), `.webp`. Maximum recommended size: **20 MB** (smaller is faster).
`mime_type`	Character or `NULL`. MIME type of the image. When `NULL` (default), auto-detected from the file extension: • `.jpg` / `.jpeg` → `"image/jpeg"` • `.png` → `"image/png"`

- `.gif` → `"image/gif"`
- `.webp` → `"image/webp"`

Override only if the extension is missing or wrong.

`detail` Character. Image detail level: `"auto"` (default), `"low"`, or `"high"`. See `image_from_url` for full explanation.

Value

A named list (content part) ready to include in a message's `content` list.

See Also

`image_from_url`, `image_from_plot`, `create_multimodal_message`

Examples

```
## Not run:
library(openaiRtools)
client <- OpenAI$new(api_key = "sk-xxxxxx")

# Send a local chart image
img <- image_from_file("results/regression_plot.png", detail = "high")

messages <- list(
  list(
    role = "user",
    content = list(
      list(
        type = "text",
        text = "This is a residuals-vs-fitted plot. Does it show heteroskedasticity?"
      ),
      img
    )
  )
)

response <- client$chat$completions$create(
  messages = messages,
  model    = "gpt-4o"
)
cat(response$choices[[1]]$message$content)

## End(Not run)
```

`image_from_plot` *Create Image Content from an R Plot Object*

Description

Renders a ggplot2 plot (or any R base graphics expression) to a temporary PNG file and encodes it as Base64, ready to be sent to a vision LLM. This lets you analyze charts produced in R without saving them manually.

Usage

```
image_from_plot(plot = NULL, width = 7, height = 5, dpi = 150, detail = "auto")
```

Arguments

plot	A ggplot object (**ggplot2**), or NULL to capture the *current* base-R graphics device (call your plot()/ hist() etc. first, then call image_from_plot(NULL)).
width	Numeric. Width of the saved PNG in inches. Default: 7.
height	Numeric. Height of the saved PNG in inches. Default: 5.
dpi	Integer. Resolution in dots per inch. Higher DPI gives sharper images (important for text readability) but larger file size. Default: 150. Use 200+ when text in plots must be legible.
detail	Character. Image detail level passed to the API: "auto" (default), "low", or "high".

Value

A named list (content part) ready to include in a message's content list.

See Also

image_from_file, create_multimodal_message

Examples

```
## Not run:
library(openaiRtools)
library(ggplot2)
client <- OpenAI$new(api_key = "sk-xxxxxx")

# Build a ggplot2 chart
p <- ggplot(mtcars, aes(x = wt, y = mpg)) +
  geom_point() +
  geom_smooth(method = "lm") +
  labs(title = "Car Weight vs Fuel Efficiency", x = "Weight", y = "MPG")

# Send plot directly to GPT-4o
response <- client$chat$completions$create(
  messages = list(
    list(
      role = "user",
      content = list(
        list(
```

```
          type = "text",
          text = "Describe the relationship shown in this scatter plot."
        ),
        image_from_plot(p, dpi = 150)
      )
    )
  ),
  model = "gpt-4o"
)
cat(response$choices[[1]]$message$content)

## End(Not run)
```

`image_from_url` *Create Image Content from a URL*

Description

Builds a content part object that tells the model to fetch and analyze an image from a public web URL. The URL must be directly accessible (no login required).

Usage

```
image_from_url(url, detail = "auto")
```

Arguments

`url`	Character. A publicly accessible image URL. Supported formats: JPEG, PNG, GIF (static), WebP. Example: `"https://example.com/chart.png"`
`detail`	Character. Controls how the model perceives the image, trading off between cost/speed and accuracy: • `"auto"` (default) — model chooses based on image size • `"low"` — 85 tokens flat; fast and cheap; good for simple images, icons, or when spatial detail is unimportant • `"high"` — tiles the image at 512px squares (up to 1105 extra tokens); use for charts with small text, detailed figures, or when precise spatial understanding is needed

Value

A named list (content part) to be placed inside a message's `content` list. Use with `create_multimodal_message` or construct messages manually.

See Also

`image_from_file`, `image_from_plot`, `create_multimodal_message`

Examples

```
## Not run:
library(openaiRtools)
client <- OpenAI$new(api_key = "sk-xxxxxx")

# Build an image part from a URL
img <- image_from_url(
  url    = "https://example.com/photo.jpg",
  detail = "low"
)

# Assemble a message manually
messages <- list(
  list(
    role = "user",
    content = list(
      list(type = "text", text = "What painting is this?"),
      img
    )
  )
)

response <- client$chat$completions$create(
  messages = messages,
  model    = "gpt-4o"
)
cat(response$choices[[1]]$message$content)

## End(Not run)
```

`ImagesClient` *Images Client*

Description

Client for OpenAI Images API (DALL-E image generation and editing). Access via `client$images`.

Methods

`$create(prompt, ...)` Generate a new image from a text description

`$edit(image, prompt, ...)` Edit an existing image using a mask

`$create_variation(image, ...)` Create variations of an existing image

Methods

Public methods:

- `ImagesClient$new()`
- `ImagesClient$create()`

- ImagesClient$edit()
- ImagesClient$create_variation()
- ImagesClient$clone()

Method `new()`**:**

Usage:

```
ImagesClient$new(parent)
```

Method `create()`**:**

Usage:

```
ImagesClient$create(
  prompt,
  model = "dall-e-3",
  n = NULL,
  quality = NULL,
  response_format = NULL,
  size = NULL,
  style = NULL,
  user = NULL
)
```

Method `edit()`**:**

Usage:

```
ImagesClient$edit(
  image,
  prompt,
  mask = NULL,
  model = "dall-e-2",
  n = NULL,
  response_format = NULL,
  size = NULL,
  user = NULL
)
```

Method `create_variation()`**:**

Usage:

```
ImagesClient$create_variation(
  image,
  model = "dall-e-2",
  n = NULL,
  response_format = NULL,
  size = NULL,
  user = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ImagesClient$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

`list_assistants` *List Assistants (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and lists all assistants.

Usage

```
list_assistants(...)
```

Arguments

`...`	Additional parameters passed to AssistantsClient`$list()`: `limit` (max results), `order` (`"asc"`/`"desc"`), `after`, `before`.

Value

A list with `$data` — a list of assistant objects, each with `$id`, `$name`, `$model`, `$instructions`, and `$tools`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

assistants <- list_assistants()
for (a in assistants$data) {
  cat(a$id, "-", a$name %||% "(unnamed)", "\n")
}

## End(Not run)
```

list_batches *List Batch Jobs (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and lists batch jobs.

Usage

```
list_batches(...)
```

Arguments

`...`	Additional parameters passed to BatchClient`$list()`, such as `limit` and `after` (pagination cursor).

Value

A list with `$data` — a list of batch objects, each containing `$id`, `$status`, `$request_counts`, and `$output_file_id`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

batches <- list_batches(limit = 5)
for (b in batches$data) cat(b$id, b$status, "\n")

## End(Not run)
```

list_files *List Uploaded Files (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$files$list()`.

Usage

```
list_files(purpose = NULL, ...)
```

Arguments

purpose	Character or NULL. Filter by purpose: "assistants", "vision", "batch", or "fine-tune". If NULL, all files are returned. Default: NULL.
...	Additional parameters passed to FilesClient$list(), such as limit, after, order.

Value

A list with $data — a list of file objects, each containing $id, $filename, $bytes, $purpose, $status, and $created_at.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# List all files
all_files <- list_files()
cat("Total files:", length(all_files$data))

# List only fine-tuning files
ft_files <- list_files(purpose = "fine-tune")
for (f in ft_files$data) cat(f$id, f$filename, "\n")

## End(Not run)
```

---

list_fine_tuning_checkpoints

List Fine-tuning Checkpoints (Convenience Function)

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and lists checkpoints for a fine-tuning job. Checkpoints are saved after each epoch and can be used as models directly.

Usage

```
list_fine_tuning_checkpoints(fine_tuning_job_id, ...)
```

Arguments

fine_tuning_job_id	Character. **Required.** The fine-tuning job ID.
...	Additional parameters passed to FineTuningCheckpointsClient$list(), such as limit and after.

Value

A list with $data — a list of checkpoint objects, each containing $fine_tuned_model_checkpoint (usable model ID), $step_number, and $metrics (list with $train_loss, $valid_loss, etc.).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

cps <- list_fine_tuning_checkpoints("ftjob-abc123")
for (cp in cps$data) {
  cat("Step:", cp$step_number, "Loss:", cp$metrics$train_loss, "\n")
}

## End(Not run)
```

---

list_fine_tuning_events

List Fine-tuning Events (Convenience Function)

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and lists training events for a fine-tuning job. Events include per-step training loss metrics and status messages.

Usage

```
list_fine_tuning_events(fine_tuning_job_id, ...)
```

Arguments

fine_tuning_job_id	Character. **Required.** The fine-tuning job ID.
...	Additional parameters passed to FineTuningJobsClient$list_events(), such as limit (max events to return) and after (pagination cursor).

Value

A list with $data — a list of event objects, each containing $message (description), $level, and $created_at (Unix timestamp).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

events <- list_fine_tuning_events("ftjob-abc123", limit = 50)
for (e in events$data) cat(e$message, "\n")

## End(Not run)
```

`list_fine_tuning_jobs` *List Fine-tuning Jobs (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and lists fine-tuning jobs.

Usage

```
list_fine_tuning_jobs(...)
```

Arguments

`...`	Additional parameters passed to FineTuningJobsClient`$list()`, such as `limit` (max jobs to return) and `after` (pagination cursor).

Value

A list with `$data` — a list of fine-tuning job objects, each containing `$id`, `$status`, `$model`, `$fine_tuned_model`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

jobs <- list_fine_tuning_jobs(limit = 5)
for (j in jobs$data) cat(j$id, "-", j$status, "\n")

## End(Not run)
```

`list_messages` *List Messages in a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and lists messages in a thread. After a run completes, the assistant's reply appears in this list.

Usage

```
list_messages(thread_id, ...)
```

Arguments

`thread_id`	Character. **Required.** The thread ID.
`...`	Additional parameters passed to MessagesClient`$list()`, such as `limit`, `order` (`"asc"`/`"desc"`), `after`, `run_id`.

Value

A list with `$data` — a list of message objects, newest first (by default). Access reply text via: `msgs$data[[1]]$content[[1]]$text$value`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Read latest messages (newest first)
msgs <- list_messages("thread_abc123")
cat("Assistant reply:", msgs$data[[1]]$content[[1]]$text$value)

# Print full conversation chronologically
msgs <- list_messages("thread_abc123", order = "asc")
for (m in msgs$data) {
  cat(toupper(m$role), ":", m$content[[1]]$text$value, "\n\n")
}

## End(Not run)
```

list_models *List Available Models (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and returns all models available to your API key.

Usage

```
list_models()
```

Value

A list with `$data` — a list of model objects, each containing `$id` (model name string), `$owned_by`, and `$created` (Unix timestamp).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

models <- list_models()

# Extract all model IDs as a character vector
model_ids <- sapply(models$data, `[[`, "id")
cat(model_ids, sep = "\n")

# Find all embedding models
embed_models <- model_ids[grepl("embedding", model_ids)]
cat(embed_models, sep = "\n")

## End(Not run)
```

list_vector_stores *List Vector Stores (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and lists all vector stores.

Usage

```
list_vector_stores(...)
```

Arguments

...	Parameters passed to VectorStoresClient$list(): limit, order, after, before.

Value

A list with $data — a list of vector store objects.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

stores <- list_vector_stores()
for (s in stores$data) cat(s$id, "-", s$name, "\n")

## End(Not run)
```

MessagesClient *Messages Client*

Description

Manages messages within threads. Messages store the conversation history between the user and the assistant. Access via client$threads$messages.

Methods

Public methods:

- MessagesClient$new()
- MessagesClient$create()
- MessagesClient$list()
- MessagesClient$retrieve()
- MessagesClient$update()
- MessagesClient$delete()
- MessagesClient$clone()

Method new():

Usage:

```
MessagesClient$new(parent)
```

Method create():

Usage:

```
MessagesClient$create(
  thread_id,
  role,
  content,
  attachments = NULL,
  metadata = NULL
)
```

Method `list()`**:**

Usage:

```
MessagesClient$list(
  thread_id,
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL,
  run_id = NULL
)
```

Method `retrieve()`**:**

Usage:

```
MessagesClient$retrieve(thread_id, message_id)
```

Method `update()`**:**

Usage:

```
MessagesClient$update(thread_id, message_id, metadata = NULL)
```

Method `delete()`**:**

Usage:

```
MessagesClient$delete(thread_id, message_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
MessagesClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`ModelsClient` *Models Client*

Description

Client for the OpenAI Models API. Allows listing available models, retrieving model details, and deleting fine-tuned models. Access via `client$models`.

Methods

`$list()` List all available models

`$retrieve(model)` Get details of a specific model

`$delete(model)` Delete a fine-tuned model you own

Methods

Public methods:

- `ModelsClient$new()`
- `ModelsClient$list()`
- `ModelsClient$retrieve()`
- `ModelsClient$delete()`
- `ModelsClient$clone()`

Method `new()`:

Usage:

```
ModelsClient$new(parent)
```

Method `list()`:

Usage:

```
ModelsClient$list()
```

Method `retrieve()`:

Usage:

```
ModelsClient$retrieve(model)
```

Method `delete()`:

Usage:

```
ModelsClient$delete(model)
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ModelsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

ModerationsClient *Moderations Client*

Description

Client for the OpenAI Moderations API. Classifies text (and optionally images) for potentially harmful content according to OpenAI's usage policies. Access via `client$moderations`.

Details

The API returns a set of category flags and confidence scores. It is free to use and does not count against your token quota.

Detected categories

`hate`, `hate/threatening`, `harassment`, `harassment/threatening`, `self-harm`, `self-harm/intent`, `self-harm/instructions`, `sexual`, `sexual/minors`, `violence`, `violence/graphic`.

Methods

Public methods:

- `ModerationsClient$new()`
- `ModerationsClient$create()`
- `ModerationsClient$clone()`

Method `new()`:

Usage:

```
ModerationsClient$new(parent)
```

Method `create()`:

Usage:

```
ModerationsClient$create(input, model = "omni-moderation-latest")
```

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
ModerationsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

multimodal *Helper Functions for Multimodal Content*

Description

Most modern LLMs (GPT-4o, Claude 3, Gemini, Qwen-VL, etc.) support multimodal input — you can send both text and images in the same message. Images are embedded inside the `content` field of a `"user"` message as a list of content parts.

There are three ways to provide an image:

1. **Image URL** — the model downloads it directly (`image_from_url`)
2. **Local file** — read and Base64-encoded automatically (`image_from_file`)
3. **R plot** — save a ggplot2 / base R figure and send it (`image_from_plot`)

Use `create_multimodal_message` to combine text + multiple images into a single ready-to-use message object.

Details

Functions to construct image content objects for sending images to vision-capable LLMs via the Chat Completions API.

OpenAI *OpenAI Client Class*

Description

Main client class for interacting with OpenAI API. Compatible with Python OpenAI SDK interface.

Methods

Public methods:

- `OpenAI$new()`
- `OpenAI$build_headers()`
- `OpenAI$request()`
- `OpenAI$request_multipart()`
- `OpenAI$request_raw()`
- `OpenAI$clone()`

Method `new()`:

Usage:

```
OpenAI$new(
  api_key = NULL,
  base_url = NULL,
  organization = NULL,
  project = NULL,
  timeout = 600,
  max_retries = 2
)
```

Method `build_headers()`**:**

Usage:

```
OpenAI$build_headers()
```

Method `request()`**:**

Usage:

```
OpenAI$request(
  method,
  path,
  body = NULL,
  query = NULL,
  stream = FALSE,
  callback = NULL
)
```

Method `request_multipart()`**:**

Usage:

```
OpenAI$request_multipart(method, path, ...)
```

Method `request_raw()`**:**

Usage:

```
OpenAI$request_raw(method, path, body = NULL)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
OpenAI$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

OpenAIAPIError *OpenAI API Error*

Description

OpenAI API Error

Usage

```
OpenAIAPIError(message, status_code = NULL, response = NULL, ...)
```

Arguments

message	Error message
status_code	HTTP status code
response	Raw response object
...	Additional arguments

OpenAIConnectionError *OpenAI Connection Error*

Description

OpenAI Connection Error

Usage

```
OpenAIConnectionError(message, ...)
```

Arguments

message	Error message
...	Additional arguments

ResponsesClient *Responses Client*

Description

The Responses API is OpenAI's next-generation API that simplifies multi-turn conversations (via `previous_response_id`), supports web search, file search, and computer use as built-in tools, and provides a cleaner interface than Chat Completions for complex agentic applications.

Details

Client for the OpenAI Responses API — a new, unified API for generating text responses, managing multi-turn conversations, and using built-in tools. Access via `client$responses`.

Methods

Public methods:

- `ResponsesClient$new()`
- `ResponsesClient$create()`
- `ResponsesClient$retrieve()`
- `ResponsesClient$delete()`
- `ResponsesClient$cancel()`
- `ResponsesClient$list_input_items()`
- `ResponsesClient$clone()`

Method `new()`:

Usage:

```
ResponsesClient$new(parent)
```

Method `create()`:

Usage:

```
ResponsesClient$create(
  model,
  input,
  instructions = NULL,
  previous_response_id = NULL,
  tools = NULL,
  tool_choice = NULL,
  parallel_tool_calls = NULL,
  max_output_tokens = NULL,
  max_completion_tokens = NULL,
  temperature = NULL,
  top_p = NULL,
  truncation = NULL,
  metadata = NULL,
```

```
  reasoning = NULL,
  service_tier = NULL,
  prompt_cache_key = NULL,
  prompt_cache_retention = NULL,
  include = NULL,
  store = NULL,
  stream = NULL,
  callback = NULL
)
```

Method `retrieve()`**:**

Usage:

```
ResponsesClient$retrieve(response_id)
```

Method `delete()`**:**

Usage:

```
ResponsesClient$delete(response_id)
```

Method `cancel()`**:**

Usage:

```
ResponsesClient$cancel(response_id)
```

Method `list_input_items()`**:**

Usage:

```
ResponsesClient$list_input_items(response_id, after = NULL, limit = NULL)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ResponsesClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`retrieve_assistant` *Retrieve an Assistant (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves a specific assistant.

Usage

```
retrieve_assistant(assistant_id)
```

Arguments

assistant_id	Character. **Required.** The assistant ID (e.g. "asst_abc123").

Value

An assistant object with $id, $name, $model, $instructions, $tools, and $created_at.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

asst <- retrieve_assistant("asst_abc123")
cat("Name:", asst$name)
cat("Model:", asst$model)

## End(Not run)
```

---

retrieve_batch *Retrieve a Batch Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and retrieves a specific batch. Use this to poll status and get the output_file_id when the batch is complete.

Usage

```
retrieve_batch(batch_id)
```

Arguments

batch_id	Character. **Required.** The batch ID (e.g. "batch_abc123").

Value

A batch object with $status, $request_counts ($total, $completed, $failed), and $output_file_id (when done).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

batch <- retrieve_batch("batch_abc123")
cat("Status:", batch$status)
if (batch$status == "completed") {
  raw <- retrieve_file_content(batch$output_file_id)
```

```
  cat(rawToChar(raw))
}

## End(Not run)
```

`retrieve_file` *Retrieve File Metadata (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves metadata for a specific file.

Usage

```
retrieve_file(file_id)
```

Arguments

`file_id`	Character. **Required.** The file ID (e.g. `"file-abc123"`).

Value

A file object with `$id`, `$filename`, `$bytes`, `$purpose`, `$status`, and `$created_at`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

info <- retrieve_file("file-abc123")
cat("Filename:", info$filename)
cat("Status:", info$status) # "processed"

## End(Not run)
```

`retrieve_file_content` *Retrieve File Content (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and downloads the raw content of a file.

Usage

```
retrieve_file_content(file_id)
```

Arguments

file_id	Character. **Required.** The file ID to download.

Value

A raw vector of binary content. Use rawToChar() for text, or writeBin() to save to disk.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Download batch output file
raw_data <- retrieve_file_content("file-abc123")
text <- rawToChar(raw_data)
cat(text)

# Save to file
writeBin(raw_data, "batch_results.jsonl")

## End(Not run)
```

---

retrieve_fine_tuning_job *Retrieve a Fine-tuning Job (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and retrieves a specific fine-tuning job.

Usage

```
retrieve_fine_tuning_job(fine_tuning_job_id)
```

Arguments

fine_tuning_job_id	Character. **Required.** The fine-tuning job ID (e.g. "ftjob-abc123").

Value

A fine-tuning job object with $status, $model, and $fine_tuned_model (the resulting model ID when status is "succeeded").

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

job <- retrieve_fine_tuning_job("ftjob-abc123")
cat("Status:", job$status)
if (job$status == "succeeded") cat("Model:", job$fine_tuned_model)

## End(Not run)
```

---

retrieve_model *Retrieve a Model (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves details for a specific model.

Usage

```
retrieve_model(model)
```

Arguments

`model`	Character. **Required.** The model ID to look up. Examples: `"gpt-4o"`, `"gpt-3.5-turbo"`, `"whisper-1"`, `"text-embedding-3-small"`, `"dall-e-3"`.

Value

A named list with model metadata: `$id`, `$object`, `$created` (Unix timestamp), `$owned_by`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

info <- retrieve_model("gpt-4o")
cat("ID:", info$id, "\n")
cat("Owner:", info$owned_by, "\n")

## End(Not run)
```

retrieve_response *Retrieve a Response (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves a stored response by its ID. Only responses created with `store = TRUE` can be retrieved.

Usage

```
retrieve_response(response_id)
```

Arguments

`response_id`	Character. **Required.** The response ID (e.g. `"resp_abc123"`).

Value

A response object with `$output`, `$model`, and `$usage`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

resp <- retrieve_response("resp_abc123")
cat(resp$output[[1]]$content[[1]]$text)

## End(Not run)
```

retrieve_run *Retrieve a Run (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves a run's current status.

Usage

```
retrieve_run(thread_id, run_id)
```

Arguments

`thread_id`	Character. **Required.** The thread ID.
`run_id`	Character. **Required.** The run ID.

Value

A run object with `$status` (`"queued"`, `"in_progress"`, `"completed"`, `"requires_action"`, `"failed"`, etc.).

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
run <- retrieve_run("thread_abc123", "run_abc123")
cat("Status:", run$status)

## End(Not run)
```

`retrieve_thread` *Retrieve a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves a thread.

Usage

```
retrieve_thread(thread_id)
```

Arguments

`thread_id`	Character. **Required.** The thread ID to retrieve.

Value

A thread object with `$id`, `$created_at`, and `$metadata`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
thread <- retrieve_thread("thread_abc123")
cat("Created at:", thread$created_at)

## End(Not run)
```

retrieve_vector_store *Retrieve a Vector Store (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and retrieves a vector store by its ID.

Usage

```
retrieve_vector_store(vector_store_id)
```

Arguments

`vector_store_id`
: Character. **Required.** The vector store ID (e.g. `"vs_abc123"`).

Value

A vector store object with `$id`, `$name`, `$status`, and `$file_counts`.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

vs <- retrieve_vector_store("vs_abc123")
cat("Name:", vs$name)
cat("Files:", vs$file_counts$completed, "ready")

## End(Not run)
```

RunsClient *Runs Client*

Description

Manages runs on threads. A run executes an assistant's logic against a thread's messages and produces a response. Access via `client$threads$runs`.

Run status lifecycle

`"queued"` → `"in_progress"` → `"completed"` (success) or → `"requires_action"` (tool call needed) → `"in_progress"` (after submitting outputs) or → `"failed"` / `"cancelled"` / `"expired"`

Methods

Public methods:

- RunsClient$new()
- RunsClient$create()
- RunsClient$list()
- RunsClient$retrieve()
- RunsClient$update()
- RunsClient$cancel()
- RunsClient$submit_tool_outputs()
- RunsClient$clone()

Method `new()`:

Usage:

```
RunsClient$new(parent)
```

Method `create()`:

Usage:

```
RunsClient$create(
  thread_id,
  assistant_id,
  model = NULL,
  instructions = NULL,
  additional_instructions = NULL,
  additional_messages = NULL,
  tools = NULL,
  metadata = NULL,
  temperature = NULL,
  top_p = NULL,
  max_prompt_tokens = NULL,
  max_completion_tokens = NULL,
  truncation_strategy = NULL,
  tool_choice = NULL,
  parallel_tool_calls = NULL,
  response_format = NULL,
  stream = NULL,
  callback = NULL
)
```

Method `list()`:

Usage:

```
RunsClient$list(
  thread_id,
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL
)
```

Method `retrieve()`**:**

Usage:

```
RunsClient$retrieve(thread_id, run_id)
```

Method `update()`**:**

Usage:

```
RunsClient$update(thread_id, run_id, metadata = NULL)
```

Method `cancel()`**:**

Usage:

```
RunsClient$cancel(thread_id, run_id)
```

Method `submit_tool_outputs()`**:**

Usage:

```
RunsClient$submit_tool_outputs(
  thread_id,
  run_id,
  tool_outputs,
  stream = NULL,
  callback = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RunsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`RunStepsClient` *Run Steps Client*

Description

Lists and retrieves the individual steps taken during a run. Each run may consist of multiple steps (e.g. thinking, tool calls, message creation). Access via `client$threads$runs$steps`.

Methods

Public methods:

- `RunStepsClient$new()`
- `RunStepsClient$list()`
- `RunStepsClient$retrieve()`
- `RunStepsClient$clone()`

Method `new()`**:**

Usage:

```
RunStepsClient$new(parent)
```

Method `list()`**:**

Usage:

```
RunStepsClient$list(
  thread_id,
  run_id,
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL
)
```

Method `retrieve()`**:**

Usage:

```
RunStepsClient$retrieve(thread_id, run_id, step_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
RunStepsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`SpeechClient` *Speech Client (Text-to-Speech)*

Description

Converts text to spoken audio using OpenAI's TTS models. Access via `client$audio$speech`.

Methods

Public methods:

- `SpeechClient$new()`
- `SpeechClient$create()`
- `SpeechClient$clone()`

Method `new()`**:**

Usage:

```
SpeechClient$new(parent)
```

Method `create()`**:**

Usage:

```
SpeechClient$create(
  input,
  model = "tts-1",
  voice = "alloy",
  response_format = NULL,
  speed = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
SpeechClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`StreamIterator` *Stream Iterator Class*

Description

R6 class for iterating over streaming response chunks.

Methods

Public methods:

- `StreamIterator$new()`
- `StreamIterator$next_chunk()`
- `StreamIterator$has_more()`
- `StreamIterator$reset()`
- `StreamIterator$as_list()`
- `StreamIterator$get_full_text()`
- `StreamIterator$clone()`

Method `new()`**:**

Usage:

```
StreamIterator$new(chunks)
```

Method `next_chunk()`**:**

Usage:

```
StreamIterator$next_chunk()
```

Method `has_more()`**:**

Usage:

```
StreamIterator$has_more()
```

Method `reset()`**:**

Usage:

```
StreamIterator$reset()
```

Method `as_list()`**:**

Usage:

```
StreamIterator$as_list()
```

Method `get_full_text()`**:**

Usage:

```
StreamIterator$get_full_text()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
StreamIterator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`text_content` *Create a Text Content Part*

Description

Wraps a plain text string into the content-part format required by the multimodal Chat Completions API. Useful when building message `content` lists manually alongside image parts.

Usage

```
text_content(text)
```

Arguments

`text`	Character. The text string to include in the message content.

Value

A named list: `list(type = "text", text = <text>)`.

See Also

`image_from_url`, `create_multimodal_message`

Examples

```
## Not run:
part <- text_content("What do you see in this image?")
# Result: list(type = "text", text = "What do you see in this image?")

## End(Not run)
```

`ThreadsClient` *Threads Client (Beta)*

Description

A Thread stores the message history of a conversation with an Assistant. Threads are persistent: they accumulate messages over multiple runs and can be retrieved at any time.

Details

Client for the OpenAI Threads API v2 (Beta). Threads are persistent conversation containers for Assistants. Access via `client$threads`.

Sub-clients

`$runs` RunsClient — Create and manage runs on threads

`$messages` MessagesClient — Add and read thread messages

Typical workflow

1. Create a thread: `client$threads$create()`
2. Add user message: `client$threads$messages$create(thread_id, "user", "...")`
3. Create a run: `client$threads$runs$create(thread_id, assistant_id)`
4. Poll run until `$status == "completed"`
5. Read response: `client$threads$messages$list(thread_id)`

Methods

Public methods:

- `ThreadsClient$new()`
- `ThreadsClient$create()`
- `ThreadsClient$retrieve()`
- `ThreadsClient$update()`
- `ThreadsClient$delete()`
- `ThreadsClient$create_and_run()`
- `ThreadsClient$clone()`

Method `new()`:

Usage:

```
ThreadsClient$new(parent)
```

Method `create()`:

Usage:

```
ThreadsClient$create(messages = NULL, tool_resources = NULL, metadata = NULL)
```

Method `retrieve()`**:**

Usage:

```
ThreadsClient$retrieve(thread_id)
```

Method `update()`**:**

Usage:

```
ThreadsClient$update(thread_id, ...)
```

Method `delete()`**:**

Usage:

```
ThreadsClient$delete(thread_id)
```

Method `create_and_run()`**:**

Usage:

```
ThreadsClient$create_and_run(
  assistant_id,
  thread = NULL,
  model = NULL,
  instructions = NULL,
  tools = NULL,
  tool_resources = NULL,
  metadata = NULL,
  stream = NULL,
  callback = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
ThreadsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`update_assistant` *Update an Assistant (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and updates an existing assistant.

Usage

```
update_assistant(assistant_id, ...)
```

Arguments

assistant_id	Character. **Required.** The ID of the assistant to update.
...	Named fields to update. Supported: name, description, instructions, model, tools, tool_resources, metadata, temperature, response_format.

Value

The updated assistant object.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

updated <- update_assistant(
  "asst_abc123",
  instructions = "You are now an expert in time series analysis.",
  model        = "gpt-4o"
)
cat("Updated model:", updated$model)

## End(Not run)
```

---

update_thread *Update a Thread (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the OPENAI_API_KEY environment variable and updates a thread's metadata.

Usage

```
update_thread(thread_id, ...)
```

Arguments

thread_id	Character. **Required.** The thread ID.
...	Named fields to update, e.g. metadata = list(label = "session-2").

Value

The updated thread object.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")
update_thread("thread_abc123", metadata = list(topic = "GMM lecture"))

## End(Not run)
```

`upload_file` *Upload a File (Convenience Function)*

Description

Shortcut that creates an OpenAI client from the `OPENAI_API_KEY` environment variable and calls `client$files$create()`.

Usage

```
upload_file(file, purpose)
```

Arguments

`file`	Character or raw. **Required.** Local file path or raw bytes.
`purpose`	Character. **Required.** Intended use of the file: `"assistants"`, `"vision"`, `"batch"`, or `"fine-tune"`.

Value

A file object with `$id` (the file ID), `$status`, `$filename`, `$bytes`, and other metadata.

Examples

```
## Not run:
Sys.setenv(OPENAI_API_KEY = "sk-xxxxxx")

# Upload training data for fine-tuning
file_obj <- upload_file("my_training_data.jsonl", purpose = "fine-tune")
cat("Uploaded file ID:", file_obj$id)

# Upload a document for an Assistant
file_obj <- upload_file("research_paper.pdf", purpose = "assistants")
cat("File ID:", file_obj$id)

## End(Not run)
```

UploadsClient *Uploads Client*

Description

The Uploads API is designed for files larger than what the Files API can handle in a single request. You split the file into parts, upload each part, then finalize the upload to get a standard file object.

Details

Client for the OpenAI Uploads API. Upload large files in multiple parts (multipart upload) when the file exceeds the 512 MB Files API limit. Access via `client$uploads`.

Workflow

1. `$create()` — Initialize the upload session, get an `upload_id`
2. `$add_part()` — Upload each chunk of the file (called multiple times)
3. `$complete()` — Finalize and assemble the file; returns a File object
4. (Optional) `$cancel()` — Abort if needed

Methods

Public methods:

- `UploadsClient$new()`
- `UploadsClient$create()`
- `UploadsClient$add_part()`
- `UploadsClient$complete()`
- `UploadsClient$cancel()`
- `UploadsClient$clone()`

Method `new()`:

Usage:

```
UploadsClient$new(parent)
```

Method `create()`:

Usage:

```
UploadsClient$create(purpose, filename, bytes, mime_type = NULL)
```

Method `add_part()`:

Usage:

```
UploadsClient$add_part(upload_id, data)
```

Method `complete()`:

Usage:

```
UploadsClient$complete(upload_id, part_ids, md5 = NULL)
```

Method `cancel()`**:**

Usage:

```
UploadsClient$cancel(upload_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
UploadsClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`VectorStoreFileBatchesClient`

Vector Store File Batches Client

Description

Add multiple files to a vector store in a single batch operation. More efficient than adding files one-by-one when loading many files. Access via `client$vector_stores$file_batches`.

Methods

Public methods:

- `VectorStoreFileBatchesClient$new()`
- `VectorStoreFileBatchesClient$create()`
- `VectorStoreFileBatchesClient$retrieve()`
- `VectorStoreFileBatchesClient$cancel()`
- `VectorStoreFileBatchesClient$list_files()`
- `VectorStoreFileBatchesClient$clone()`

Method `new()`**:**

Usage:

```
VectorStoreFileBatchesClient$new(parent)
```

Method `create()`**:**

Usage:

```
VectorStoreFileBatchesClient$create(
  vector_store_id,
  file_ids,
  chunking_strategy = NULL
)
```

Method `retrieve()`**:**

Usage:

```
VectorStoreFileBatchesClient$retrieve(vector_store_id, batch_id)
```

Method `cancel()`**:**

Usage:

```
VectorStoreFileBatchesClient$cancel(vector_store_id, batch_id)
```

Method `list_files()`**:**

Usage:

```
VectorStoreFileBatchesClient$list_files(
  vector_store_id,
  batch_id,
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VectorStoreFileBatchesClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`VectorStoreFilesClient` *Vector Store Files Client*

Description

Manages individual files within a vector store. Adding a file triggers automatic chunking and embedding. Access via `client$vector_stores$files`.

Methods

Public methods:

- `VectorStoreFilesClient$new()`
- `VectorStoreFilesClient$create()`
- `VectorStoreFilesClient$list()`
- `VectorStoreFilesClient$retrieve()`
- `VectorStoreFilesClient$update()`
- `VectorStoreFilesClient$delete()`
- `VectorStoreFilesClient$content()`
- `VectorStoreFilesClient$clone()`

Method `new()`**:**

Usage:

```
VectorStoreFilesClient$new(parent)
```

Method `create()`**:**

Usage:

```
VectorStoreFilesClient$create(
  vector_store_id,
  file_id,
  chunking_strategy = NULL
)
```

Method `list()`**:**

Usage:

```
VectorStoreFilesClient$list(
  vector_store_id,
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL,
  filter = NULL
)
```

Method `retrieve()`**:**

Usage:

```
VectorStoreFilesClient$retrieve(vector_store_id, file_id)
```

Method `update()`**:**

Usage:

```
VectorStoreFilesClient$update(vector_store_id, file_id, attributes = NULL)
```

Method `delete()`**:**

Usage:

```
VectorStoreFilesClient$delete(vector_store_id, file_id)
```

Method `content()`**:**

Usage:

```
VectorStoreFilesClient$content(vector_store_id, file_id)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VectorStoreFilesClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

VectorStoresClient *Vector Stores Client (Beta)*

Description

A vector store automatically chunks, embeds, and indexes files so that an Assistant with the `file_search` tool can search over them using natural language queries.

Details

Client for the OpenAI Vector Stores API v2 (Beta). Vector stores enable semantic file search for Assistants. Access via `client$vector_stores`.

Sub-clients

`$files` VectorStoreFilesClient — Add/remove files from a store

`$file_batches` VectorStoreFileBatchesClient — Batch-add files

Typical workflow

1. Upload files: `client$files$create(file, purpose = "assistants")`
2. Create vector store: `client$vector_stores$create(name = "...")`
3. Add files: `client$vector_stores$files$create(store_id, file_id)`
4. Attach to assistant via `tool_resources` in `client$assistants$create()`

Methods

Public methods:

- `VectorStoresClient$new()`
- `VectorStoresClient$create()`
- `VectorStoresClient$list()`
- `VectorStoresClient$retrieve()`
- `VectorStoresClient$update()`
- `VectorStoresClient$delete()`
- `VectorStoresClient$search()`
- `VectorStoresClient$clone()`

Method `new()`:

Usage:

```
VectorStoresClient$new(parent)
```

Method `create()`:

Usage:

```
VectorStoresClient$create(
  name = NULL,
  file_ids = NULL,
  expires_after = NULL,
  chunking_strategy = NULL,
  metadata = NULL
)
```

Method `list()`**:**

Usage:

```
VectorStoresClient$list(
  limit = NULL,
  order = NULL,
  after = NULL,
  before = NULL
)
```

Method `retrieve()`**:**

Usage:

```
VectorStoresClient$retrieve(vector_store_id)
```

Method `update()`**:**

Usage:

```
VectorStoresClient$update(vector_store_id, ...)
```

Method `delete()`**:**

Usage:

```
VectorStoresClient$delete(vector_store_id)
```

Method `search()`**:**

Usage:

```
VectorStoresClient$search(
  vector_store_id,
  query,
  filter = NULL,
  max_num_results = NULL,
  ranking_options = NULL,
  rewrite_query = NULL
)
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
VectorStoresClient$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Index