Image generation

Learn how to generate or manipulate images with DALL·E in the API.

> Looking to generate images in ChatGPT? Head to chatgpt.com.

Introduction

The Images API provides three methods for interacting with images:

1. Creating images from scratch based on a text prompt (DALL·E 3 and DALL·E 2)
2. Creating edited versions of images by having the model replace some areas of a pre-existing image, based on a new text prompt (DALL·E 2 only)
3. Creating variations of an existing image (DALL·E 2 only)

This guide covers the basics of using these three API endpoints with useful code samples. To try DALL·E 3, head to ChatGPT. To try DALL·E 2, check out the DALL·E preview app.

Usage

Generations

The image generations endpoint allows you to create an original image given a text prompt. When using DALL·E 3, images can have a size of 1024×1024, 1024×1792 or 1792×1024 pixels.

By default, images are generated at `standard` quality, but when using DALL·E 3 you can set `quality: "hd"` for enhanced detail. Square, standard quality images are the fastest to generate.

You can request 1 image at a time with DALL·E 3 (request more by making parallel requests) or up to 10 images at a time using DALL·E 2 with the n parameter.

Generate an image — python

```python
from openai import OpenAI
client = OpenAI()

response = client.images.generate(
  model="dall-e-3",
  prompt="a white siamese cat",
```

```
    size="1024x1024",
    quality="standard",
    n=1,
)

image_url = response.data[0].url
```

What is new with DALL·E 3

Explore what is new with DALL·E 3 in the OpenAI Cookbook

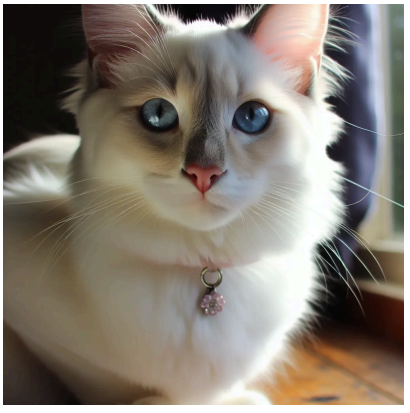
Prompting

With the release of DALL·E 3, the model now takes in the default prompt provided and automatically re-write it for safety reasons, and to add more detail (more detailed prompts generally result in higher quality images).

While it is not currently possible to disable this feature, you can use prompting to get outputs closer to your requested image by adding the following to your prompt: `I NEED to test how the tool works with extremely simple prompts. DO NOT add any detail, just use it AS-IS:`.

The updated prompt is visible in the `revised_prompt` field of the data response object.

Example DALL·E 3 generations

PROMPT	GENERATION
A photograph of a white Siamese cat.	

Each image can be returned as either a URL or Base64 data, using the response_format parameter. URLs will expire after an hour.

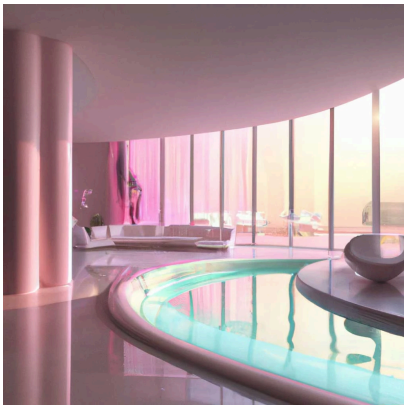
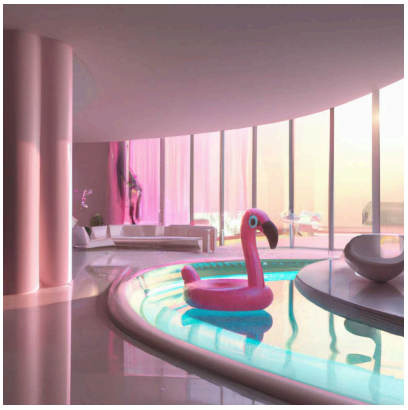
Edits (DALL·E 2 only)

Also known as "inpainting", the image edits endpoint allows you to edit or extend an image by uploading an image and mask indicating which areas should be replaced. The transparent areas of the mask indicate where the image should be edited, and the prompt should describe the full new image, **not just the erased area**. This endpoint can enable experiences like the editor in our DALL·E preview app.

Edit an image — python

```python
from openai import OpenAI
client = OpenAI()

response = client.images.edit((
  model="dall-e-2",
  image=open("sunlit_lounge.png", "rb"),
  mask=open("mask.png", "rb"),
  prompt="A sunlit indoor lounge area with a pool containing a flamingo",
  n=1,
  size="1024×1024"
)
image_url = response.data[0].url
```

IMAGE	MASK	OUTPUT
		

Prompt: a sunlit indoor lounge area with a pool containing a flamingo

The uploaded image and mask must both be square PNG images less than 4MB in size, and also must have the same dimensions as each other. The non-transparent areas of the mask are not used when generating the output, so they don't necessarily need to match the original image like the example above.

Variations (DALL·E 2 only)

The image variations endpoint allows you to generate a variation of a given image.

Generate an image variation

python

```python
from openai import OpenAI
client = OpenAI()

response = client.images.create_variation(
  model="dall-e-2",
  image=open("corgi_and_cat_paw.png", "rb"),
  n=1,
  size="1024x1024"
)

image_url = response.data[0].url
```

IMAGE	OUTPUT
	

Similar to the edits endpoint, the input image must be a square PNG image less than 4MB in size.

Content moderation

Prompts and images are filtered based on our content policy, returning an error when a prompt or image is flagged.

Language-specific tips

Node.js | Python

Using in-memory image data

The Node.js examples in the guide above use the `fs` module to read image data from disk. In some cases, you may have your image data in memory instead. Here's an example API call that uses image

data stored in a Node.js `Buffer` object:

```
import OpenAI from "openai";

const openai = new OpenAI();

// This is the Buffer object that contains your image data
const buffer = [your image data];

// Set a `name` that ends with .png so that the API knows it's a PNG image
buffer.name = "image.png";

async function main() {
  const image = await openai.images.createVariation({ model: "dall-e-2", im
  console.log(image.data);
}
main();
```

Working with TypeScript

If you're using TypeScript, you may encounter some quirks with image file arguments. Here's an example of working around the type mismatch by explicitly casting the argument:

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

const openai = new OpenAI();

async function main() {
  // Cast the ReadStream to `any` to appease the TypeScript compiler
  const image = await openai.images.createVariation({
    image: fs.createReadStream("image.png") as any,
  });

  console.log(image.data);
}
main();
```

And here's a similar example for in-memory image data:

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

const openai = new OpenAI();

// This is the Buffer object that contains your image data
const buffer: Buffer = [your image data];

// Cast the buffer to `any` so that we can set the `name` property
const file: any = buffer;

// Set a `name` that ends with .png so that the API knows it's a PNG image
file.name = "image.png";

async function main() {
  const image = await openai.images.createVariation({
    file,
    1,
    "1024x1024"
  });
  console.log(image.data);
}
main();
```

Error handling

API requests can potentially return errors due to invalid inputs, rate limits, or other issues. These errors can be handled with a `try...catch` statement, and the error details can be found in either `error.response` or `error.message`:

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

const openai = new OpenAI();

try {
```

```
    const response = openai.images.createVariation(
        fs.createReadStream("image.png"),
        1,
        "1024x1024"
    );
    console.log(response.data.data[0].url);
} catch (error) {
    if (error.response) {
        console.log(error.response.status);
        console.log(error.response.data);
    } else {
        console.log(error.message);
    }
}
```