Image generation

Copy page

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

Introduction

The Images API has three endpoints with different abilities:

Generations: Images from scratch, based on a text prompt

Edits: Edited versions of images, where the model replaces some areas of a pre-existing image, based on a new text prompt

Variations: Variations of an existing image

This guide covers the basics of using these endpoints with code samples. To try DALL·E 3 without writing any code, go to ChatGPT.

Which model should I use?

DALL·E 2 and DALL·E 3 have different options for generating images.

MODEL	AVAILABLE ENDPOINTS	BEST FOR
DALL·E 2	Generations, edits, variations	More options (edits and variations), more control in prompting, more requests at once
DALL·E 3	Only image generations	Higher quality, larger sizes for generated images

Generations

The image generations endpoint allows you to create an original image with a text prompt. Each image can be returned either as a URL or Base64 data, using the response_format parameter. The default output is URL, and each URL expires after an hour.

Size and quality options

Square, standard quality images are the fastest to generate. The default size of generated images is `1024x1024` pixels, but each model has different options:

MODEL	SIZES OPTIONS (PIXELS)	QUALITY OPTIONS	REQUESTS YOU CAN MAKE

DALL·E 2	`256x256`
	`512x512`
	`1024x1024`
		Only `standard`	Up to 10 images at a time, with the n parameter

DALL·E 3	`1024x1024`
	`1024x1792`
	`1792x1024`
		Defaults to `standard`
		Set `quality: "hd"` for enhanced detail
			Only 1 at a time, but can request more by making parallel requests

The following code example uses DALL·E 3 to generate a square, standard quality image of a cat.

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,
)

print(response.data[0].url)
```

DALL·E 3 prompting

With the release of DALL·E 3, the model takes in your prompt and automatically rewrites it:

- For safety reasons
- To add more detail (more detailed prompts generally result in higher quality images)

You can't disable this feature, but you can 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.

What's new with DALL·E 3

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

Edits (DALL·E 2 only)

The image edits endpoint lets you edit or extend an image by uploading an image and mask indicating which areas should be replaced. This process is also known as **inpainting**.

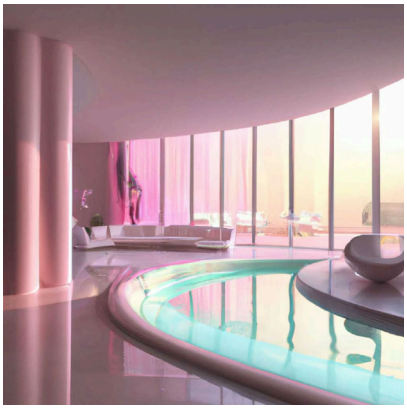
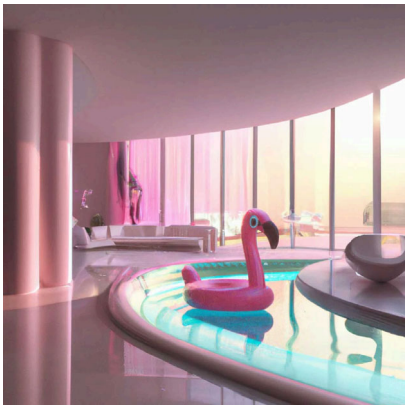
The transparent areas of the mask indicate where the image should be edited, and the prompt should describe the full new image, **not just the erased area**. This endpoint enables experiences like DALL·E image editing in ChatGPT Plus.

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="1024x1024",
)

print(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 have the same dimensions as each other. The non-transparent areas of the mask aren't used to generate the output, so they don't need to match the original image like our example.

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"
)

print(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:

```javascript
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", image: buffer, n: 1
  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();

async function main() {
    try {
        const image = await openai.images.createVariation({
            image: fs.createReadStream("image.png"),
            n: 1,
            size: "1024x1024",
        });
        console.log(image.data);
    } catch (error) {
        if (error.response) {
            console.log(error.response.status);
            console.log(error.response.data);
        } else {
            console.log(error.message);
        }
    }
}

main();
```