Image generation

Learn how to generate or manipulate images with our DALL·E models

Introduction

The Images API provides three methods for interacting with images:

1. Creating images from scratch based on a text prompt
2. Creating edits of an existing image based on a new text prompt
3. Creating variations of an existing image

This guide covers the basics of using these three API endpoints with useful code samples. To see them in action, check out our DALL·E preview app.

Usage

Generations

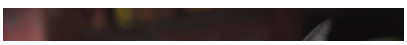
The image generations endpoint allows you to create an original image given a text prompt. Generated images can have a size of 256x256, 512x512, or 1024x1024 pixels. Smaller sizes are faster to generate. You can request 1-10 images at a time using the n parameter.

Generate an image

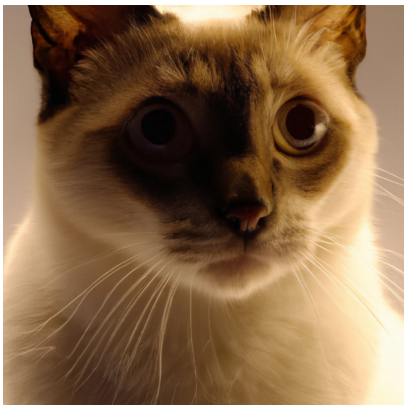
python Copy

```python
response = openai.Image.create(
  prompt="a white siamese cat",
  n=1,
  size="1024x1024"
)
image_url = response['data'][0]['url']
```

The more detailed the description, the more likely you are to get the result that you or your end user want. You can explore the examples in the DALL·E preview app for more prompting inspiration. Here's a quick example:

PROMPT	GENERATION
a white siamese cat	

a close up, studio photographic portrait of a white siamese cat that looks curious, backlit ears



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

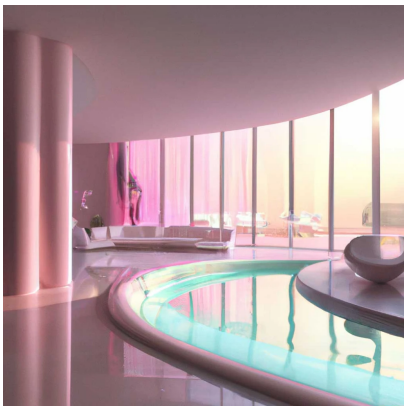
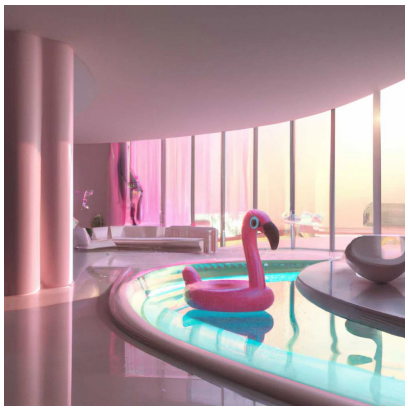
Edits

The image edits endpoint allows you to edit and extend an image by uploading a mask. 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 Copy

```python
response = openai.Image.create_edit(
  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"
)
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

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

Generate an image variation

python Copy

```python
response = openai.Image.create_variation(
  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. If you have any feedback on false positives or related issues, please contact us through our help center.

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:

```
// 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 i
buffer.name = "image.png";
const response = await openai.createImageVariation(
  buffer,
  1,
  "1024x1024"
);
```

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:

```
// Cast the ReadStream to `any` to appease the TypeScript compil
const response = await openai.createImageVariation(
```

```
  fs.createReadStream("image.png") as any,
  1,
  "1024x1024"
);
```

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

```
// 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
file.name = "image.png";
const response = await openai.createImageVariation(
  file,
  1,
  "1024x1024"
);
```

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`:

```
try {
  const response = await openai.createImageVariation(
    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);
  }
}
```