OpenAI Platform

Images and vision

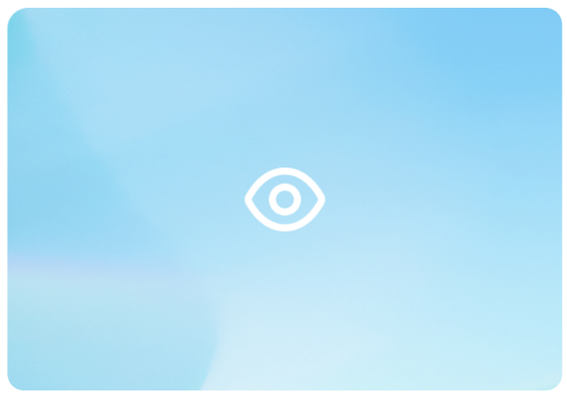
Learn how to understand or generate images.

Copy page

Responses

Overview

Create images
Use GPT Image or DALL·E to generate or edit images.



Process image inputs
Use our models' vision capabilities to analyze images.

In this guide, you will learn about building applications involving images with the OpenAI API. If you know what you want to build, find your use case below to get started. If you're not sure where to start, continue reading to get an overview.

A tour of image-related use cases

Recent language models can process image inputs and analyze them — a capability known as **vision**. With `gpt-image-1`, they can both analyze visual inputs and create images.

The OpenAI API offers several endpoints to process images as input or generate them as output, enabling you to build powerful multimodal applications.

API	SUPPORTED USE CASES
Responses API	Analyze images and use them as input and/or generate images as output
Images API	Generate images as output, optionally using images as input
Chat Completions API	Analyze images and use them as input to generate text or audio

To learn more about the input and output modalities supported by our models, refer to our models page.

Generate or edit images

You can generate or edit images using the Image API or the Responses API.

Our latest image generation model, `gpt-image-1`, is a natively multimodal large language model. It can understand text and images and leverage its broad world knowledge to generate images with better instruction following and contextual awareness.

In contrast, we also offer specialized image generation models - DALL·E 2 and 3 - which don't have the same inherent understanding of the world as GPT Image.

Generate images with Responses — python

```python
from openai import OpenAI
import base64

client = OpenAI()

response = client.responses.create(
    model="gpt-4.1-mini",
    input="Generate an image of gray tabby cat hugging an otter with an orange s
    tools=[{"type": "image_generation"}],
)

// Save the image to a file
image_data = [
    output.result
    for output in response.output
    if output.type == "image_generation_call"
]

if image_data:
    image_base64 = image_data[0]
    with open("cat_and_otter.png", "wb") as f:
        f.write(base64.b64decode(image_base64))
```

You can learn more about image generation in our Image generation guide.

Using world knowledge for image generation

The difference between DALL·E models and GPT Image is that a natively multimodal language model can use its visual understanding of the world to generate lifelike images including real-life details without a reference.

For example, if you prompt GPT Image to generate an image of a glass cabinet with the most popular semi-precious stones, the model knows enough to select gemstones like amethyst, rose

quartz, jade, etc, and depict them in a realistic way.

Analyze images

Vision is the ability for a model to "see" and understand images. If there is text in an image, the model can also understand the text. It can understand most visual elements, including objects, shapes, colors, and textures, even if there are some limitations.

Giving a model images as input

You can provide images as input to generation requests in multiple ways:

- By providing a fully qualified URL to an image file
- By providing an image as a Base64-encoded data URL
- By providing a file ID (created with the Files API)

You can provide multiple images as input in a single request by including multiple images in the `content` array, but keep in mind that images count as tokens and will be billed accordingly.

Passing a URL | Passing a Base64 encoded image | Passing a file ID

Analyze the content of an image — python

```python
from openai import OpenAI

client = OpenAI()

response = client.responses.create(
    model="gpt-4.1-mini",
    input=[{
        "role": "user",
        "content": [
            {"type": "input_text", "text": "what's in this image?"},
            {
                "type": "input_image",
                "image_url": "https://upload.wikimedia.org/wikipedia/commons/th
            },
        ],
    }],
)

print(response.output_text)
```

Image input requirements

Input images must meet the following requirements to be used in the API.

Supported file types	PNG (.png) - JPEG (.jpeg and .jpg) - WEBP (.webp) - Non-animated GIF (.gif)
Size limits	Up to 50 MB total payload size per request - Up to 500 individual image inputs per request
Other requirements	No watermarks or logos - No NSFW content - Clear enough for a human to understand

Specify image input detail level

The `detail` parameter tells the model what level of detail to use when processing and understanding the image (`low` , `high` , or `auto` to let the model decide). If you skip the parameter, the model will use `auto` .

```
{
    "type": "input_image",
    "image_url": "https://upload.wikimedia.org/wikipedia/commons/thumb/d/dd/Gfp-
    "detail": "high"
}
```

You can save tokens and speed up responses by using `"detail": "low"` . This lets the model process the image with a budget of 85 tokens. The model receives a low-resolution 512px x 512px version of the image. This is fine if your use case doesn't require the model to see with high-resolution detail (for example, if you're asking about the dominant shape or color in the image).

On the other hand, you can use `"detail": "high"` if you want the model to have a better understanding of the image.

Read more about calculating image processing costs in the Calculating costs section below.

Limitations

While models with vision capabilities are powerful and can be used in many situations, it's important to understand the limitations of these models. Here are some known limitations:

- **Medical images**: The model is not suitable for interpreting specialized medical images like CT scans and shouldn't be used for medical advice.
- **Non-English**: The model may not perform optimally when handling images with text of non-Latin alphabets, such as Japanese or Korean.
- **Small text**: Enlarge text within the image to improve readability, but avoid cropping important details.

Rotation: The model may misinterpret rotated or upside-down text and images.

Visual elements: The model may struggle to understand graphs or text where colors or styles—like solid, dashed, or dotted lines—vary.

Spatial reasoning: The model struggles with tasks requiring precise spatial localization, such as identifying chess positions.

Accuracy: The model may generate incorrect descriptions or captions in certain scenarios.

Image shape: The model struggles with panoramic and fisheye images.

Metadata and resizing: The model doesn't process original file names or metadata, and images are resized before analysis, affecting their original dimensions.

Counting: The model may give approximate counts for objects in images.

CAPTCHAS: For safety reasons, our system blocks the submission of CAPTCHAs.

Calculating costs

Image inputs are metered and charged in tokens, just as text inputs are. How images are converted to text token inputs varies based on the model. You can find a vision pricing calculator in the FAQ section of the pricing page.

GPT-4.1-mini, GPT-4.1-nano, o4-mini

Image inputs are metered and charged in tokens based on their dimensions. The token cost of an image is determined as follows:

A. Calculate the number of 32px x 32px patches that are needed to fully cover the image (a patch may extend beyond the image boundaries; out-of-bounds pixels are treated as black.)

```
raw_patches = ceil(width/32)×ceil(height/32)
```

B. If the number of patches exceeds 1536, we scale down the image so that it can be covered by no more than 1536 patches

```
r = √(32²×1536/(width×height))
r = r × min( floor(width×r/32) / (width×r/32), floor(height×r/32) / (height×r/32) )
```

C. The token cost is the number of patches, capped at a maximum of 1536 tokens

```
image_tokens = ceil(resized_width/32)×ceil(resized_height/32)
```

D. Apply a multiplier based on the model to get the total tokens.

MODEL	MULTIPLIER
`gpt-5-mini`	1.62
`gpt-5-nano`	2.46
`gpt-4.1-mini`	1.62
`gpt-4.1-nano`	2.46
`o4-mini`	1.72

Cost calculation examples

A 1024 x 1024 image is **1024 tokens**

Width is 1024, resulting in `(1024 + 32 - 1) // 32 = 32` patches

Height is 1024, resulting in `(1024 + 32 - 1) // 32 = 32` patches

Tokens calculated as `32 * 32 = 1024`, below the cap of 1536

A 1800 x 2400 image is **1452 tokens**

Width is 1800, resulting in `(1800 + 32 - 1) // 32 = 57` patches

Height is 2400, resulting in `(2400 + 32 - 1) // 32 = 75` patches

We need `57 * 75 = 4275` patches to cover the full image. Since that exceeds 1536, we need to scale down the image while preserving the aspect ratio.

We can calculate the shrink factor as `sqrt(token_budget × patch_size^2 / (width * height))`. In our example, the shrink factor is `sqrt(1536 * 32^2 / (1800 * 2400)) = 0.603`.

Width is now 1086, resulting in `1086 / 32 = 33.94` patches

Height is now 1448, resulting in `1448 / 32 = 45.25` patches

We want to make sure the image fits in a whole number of patches. In this case we scale again by `33 / 33.94 = 0.97` to fit the width in 33 patches.

The final width is then `1086 * (33 / 33.94) = 1056)` and the final height is `1448 * (33 / 33.94) = 1408`

The image now requires `1056 / 32 = 33` patches to cover the width and `1408 / 32 = 44` patches to cover the height

The total number of tokens is the `33 * 44 = 1452`, below the cap of 1536

GPT 4o, GPT-4.1, GPT-4o-mini, CUA, and o-series (except o4-mini)

The token cost of an image is determined by two factors: size and detail.

Any image with `"detail": "low"` costs a set, base number of tokens. This amount varies by model (see chart below). To calculate the cost of an image with `"detail": "high"`, we do the following:

Scale to fit in a 2048px x 2048px square, maintaining original aspect ratio

Scale so that the image's shortest side is 768px long

Count the number of 512px squares in the image—each square costs a set amount of tokens (see chart below)

Add the base tokens to the total

MODEL	BASE TOKENS	TILE TOKENS
gpt-5, gpt-5-chat-latest	70	140
4o, 4.1, 4.5	85	170
4o-mini	2833	5667
o1, o1-pro, o3	75	150
computer-use-preview	65	129

Cost calculation examples (for gpt-4o)

A 1024 x 1024 square image in `"detail": "high"` mode costs 765 tokens

1024 is less than 2048, so there is no initial resize.

The shortest side is 1024, so we scale the image down to 768 x 768.

4 512px square tiles are needed to represent the image, so the final token cost is `170 * 4 + 85 = 765`.

A 2048 x 4096 image in `"detail": "high"` mode costs 1105 tokens

We scale down the image to 1024 x 2048 to fit within the 2048 square.

The shortest side is 1024, so we further scale down to 768 x 1536.

6 512px tiles are needed, so the final token cost is `170 * 6 + 85 = 1105`.

A 4096 x 8192 image in `"detail": "low"` most costs 85 tokens

Regardless of input size, low detail images are a fixed cost.

GPT Image 1

For GPT Image 1, we calculate the cost of an image input the same way as described above, except that we scale down the image so that the shortest side is 512px instead of 768px. The price depends on the dimensions of the image and the input fidelity.

When input fidelity is set to low, the base cost is 65 image tokens, and each tile costs 129 image tokens. When using high input fidelity, we add a set number of tokens based on the image's aspect ratio in addition to the image tokens described above.

If your image is square, we add 4160 extra input image tokens.

If it is closer to portrait or landscape, we add 6240 extra tokens.

To see pricing for image input tokens, refer to our pricing page.

We process images at the token level, so each image we process counts towards your tokens per minute (TPM) limit.

For the most precise and up-to-date estimates for image processing, please use our image pricing calculator available here.