Search CTRL K

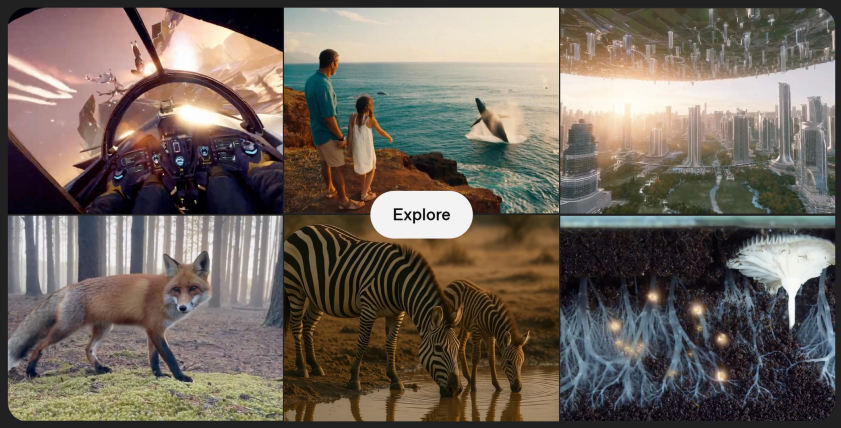
Graders

Specialized models
Image generation
Video generation
Text to speech
Speech to text
Deep research
Embeddings
Moderation

Video generation with Sora

Copy page

Create, iterate on, and manage videos with the Sora API.



Overview

Sora is OpenAI's newest frontier in generative media – a state-of-the-art video model capable of creating richly detailed, dynamic clips with audio from natural language or images. Built on years of research into multimodal diffusion and trained on diverse visual data, Sora brings a deep understanding of 3D space, motion, and scene continuity to text-to-video generation.

The Video API (in preview) exposes these capabilities to developers for the first time, enabling programmatic creation, extension, and remixing of videos. It provides five endpoints, each with distinct capabilities:

- **Create video**: Start a new render job from a prompt, with optional reference inputs or a remix ID.
- **Get video status**: Retrieve the current state of a render job and monitor its progress.
- **Download video**: Fetch the finished MP4 once the job is completed.
- **List videos**: Enumerate your videos with pagination for history, dashboards, or housekeeping.
- **Delete videos**: Remove an individual video ID from OpenAI's storage.

Models

The second generation Sora model comes in two variants, each tailored for different use cases.

Sora 2

`sora-2` is designed for **speed and flexibility**. It's ideal for the exploration phase, when you're experimenting with tone, structure, or visual style and need quick feedback rather than perfect fidelity.

It generates good quality results quickly, making it well suited for rapid iteration, concepting, and rough cuts. `sora-2` is often more than sufficient for social media content, prototypes, and scenarios where turnaround time matters more than ultra-high fidelity.

Sora 2 Pro

`sora-2-pro` produces higher quality results. It's the better choice when you need **production-quality output**.

`sora-2-pro` takes longer to render and is more expensive to run, but it produces more polished, stable results. It's best for high-resolution cinematic footage, marketing assets, and any situation where visual precision is critical.

Generate a video

Generating a video is an **asynchronous** process:

1. When you call the `POST /videos` endpoint, the API returns a job object with a job `id` and an initial `status`.
2. You can either poll the `GET /videos/{video_id}` endpoint until the status transitions to completed, or – for a more efficient approach – use webhooks (see the webhooks section below) to be notified automatically when the job finishes.
3. Once the job has reached the `completed` state you can fetch the final MP4 file with `GET /videos/{video_id}/content`.

Start a render job

Start by calling `POST /videos` with a text prompt and the required parameters. The prompt defines the creative look and feel – subjects, camera, lighting, and motion – while parameters like `size` and `seconds` control the video's resolution and length.

Create a video — javascript

```javascript
import OpenAI from 'openai';

const openai = new OpenAI();

let video = await openai.videos.create({
    model: 'sora-2',
    prompt: "A video of the words 'Thank you' in sparkling letters",
});

console.log('Video generation started: ', video);
```

The response is a JSON object with a unique id and an initial status such as `queued` or `in_progress`. This means the render job has started.

```json
{
  "id": "video_68d7512d07848190b3e45da0ecbebcde004da08e1e0678d5",
  "object": "video",
  "created_at": 1758941485,
  "status": "queued",
  "model": "sora-2-pro",
  "progress": 0,
  "seconds": "8",
  "size": "1280x720"
}
```

Guardrails and restrictions

The API enforces several content restrictions:

- Only content suitable for audiences under 18 (a setting to bypass this restriction will be available in the future).
- Copyrighted characters and copyrighted music will be rejected.
- Real people—including public figures—cannot be generated.
- Input images with faces of humans are currently rejected.

Make sure prompts, reference images, and transcripts respect these rules to avoid failed generations.

Effective prompting

For best results, describe **shot type, subject, action, setting, and lighting**. For example:

- *"Wide shot of a child flying a red kite in a grassy park, golden hour sunlight, camera slowly pans upward."*
- *"Close-up of a steaming coffee cup on a wooden table, morning light through blinds, soft depth of field."*

This level of specificity helps the model produce consistent results without inventing unwanted details. For more advanced prompting techniques, please refer to our dedicated Sora 2 prompting guide.

Monitor progress

Video generation takes time. Depending on model, API load and resolution, **a single render may**

take several minutes.

To manage this efficiently, you can poll the API to request status updates or you can get notified via a webhook.

Poll the status endpoint

Call `GET /videos/{video_id}` with the id returned from the create call. The response shows the job's current status, progress percentage (if available), and any errors.

Typical states are `queued`, `in_progress`, `completed`, and `failed`. Poll at a reasonable interval (for example, every 10–20 seconds), use exponential backoff if necessary, and provide feedback to users that the job is still in progress.

Poll the status endpoint — javascript

```javascript
import OpenAI from 'openai';

const openai = new OpenAI();

async function main() {
  const video = await openai.videos.createAndPoll({
    model: 'sora-2',
    prompt: "A video of the words 'Thank you' in sparkling letters",
  });

  if (video.status === 'completed') {
    console.log('Video successfully completed: ', video);
  } else {
    console.log('Video creation failed. Status: ', video.status);
  }
}

main();
```

Response example:

```
{
  "id": "video_68d7512d07848190b3e45da0ecbebcde004da08e1e0678d5",
  "object": "video",
  "created_at": 1758941485,
  "status": "in_progress",
  "model": "sora-2-pro",
  "progress": 33,
  "seconds": "8",
  "size": "1280x720"
}
```

Use webhooks for notifications

Instead of polling job status repeatedly with `GET`, register a webhook to be notified automatically when a video generation completes or fails.

Webhooks can be configured in your webhook settings page. When a job finishes, the API emits one of two event types: `video.completed` and `video.failed`. Each event includes the ID of the job that triggered it.

Example webhook payload:

```
{
  "id": "evt_abc123",
  "object": "event",
  "created_at": 1758941485,
  "type": "video.completed", // or "video.failed"
  "data": {
    "id": "video_abc123"
  }
}
```

Retrieve results

Download the MP4

Once the job reaches status `completed`, fetch the MP4 with `GET /videos/{video_id}/content`. This endpoint streams the binary video data and returns standard content headers, so you can

either save the file directly to disk or pipe it to cloud storage.

Download the MP4 javascript

```javascript
import OpenAI from 'openai';

const openai = new OpenAI();

let video = await openai.videos.create({
    model: 'sora-2',
    prompt: "A video of the words 'Thank you' in sparkling letters",
});

console.log('Video generation started: ', video);
let progress = video.progress ?? 0;

while (video.status === 'in_progress' || video.status === 'queued') {
    video = await openai.videos.retrieve(video.id);
    progress = video.progress ?? 0;

    // Display progress bar
    const barLength = 30;
    const filledLength = Math.floor((progress / 100) * barLength);
    // Simple ASCII progress visualization for terminal output
    const bar = '='.repeat(filledLength) + '-'.repeat(barLength - filledLength);
    const statusText = video.status === 'queued' ? 'Queued' : 'Processing';

    process.stdout.write(`${statusText}: [${bar}] ${progress.toFixed(1)}%`);

    await new Promise((resolve) => setTimeout(resolve, 2000));
}

// Clear the progress line and show completion
process.stdout.write('\n');

if (video.status === 'failed') {
    console.error('Video generation failed');
    return;
}

console.log('Video generation completed: ', video);

console.log('Downloading video content...');

const content = await openai.videos.downloadContent(video.id);

const body = content.arrayBuffer();
const buffer = Buffer.from(await body);

require('fs').writeFileSync('video.mp4', buffer);

console.log('Wrote video.mp4');
```

You now have the final video file ready for playback, editing, or distribution. Download URLs are valid for a maximum of 1 hour after generation. If you need long-term storage, copy the file to your own storage system promptly.

Download supporting assets

For each completed video, you can also download a **thumbnail** and a **spritesheet**. These are lightweight assets useful for previews, scrubbers, or catalog displays. Use the `variant` query parameter to specify what you want to download. The default is `variant=video` for the MP4.

```bash
# Download a thumbnail
curl -L "https://api.openai.com/v1/videos/video_abc123/content?variant=thumbnail" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  --output thumbnail.webp

# Download a spritesheet
curl -L "https://api.openai.com/v1/videos/video_abc123/content?variant=spritesheet" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  --output spritesheet.jpg
```

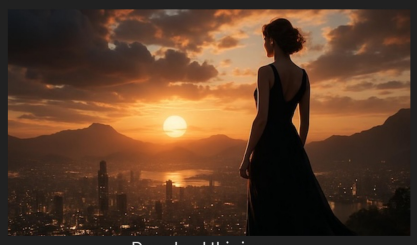
Use image references

You can guide a generation with an input image, which acts as **the first frame of your video**. This is useful if you need the output video to preserve the look of a brand asset, a character, or a specific environment. Include an image file as the `input_reference` parameter in your

specific environment. Include an image file as the `input_reference` parameter in your `POST /videos` request. The image must match the target video's resolution (`size`).

Supported file formats are `image/jpeg` , `image/png` , and `image/webp` .

```
curl -X POST "https://api.openai.com/v1/videos" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: multipart/form-data" \
  -F prompt="She turns around and smiles, then slowly walks out of the frame." \
  -F model="sora-2-pro" \
  -F size="1280x720" \
  -F seconds="8" \
  -F input_reference="@sample_720p.jpeg;type=image/jpeg"
```

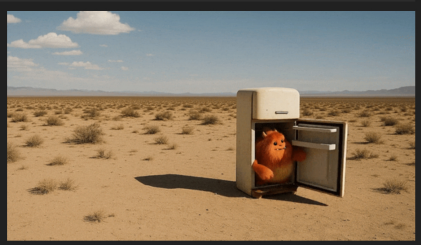
INPUT IMAGE GENERATED WITH OPENAI GPT IMAGE	GENERATED VIDEO USING SORA 2 (CONVERTED TO GIF)
 Download this image	 Prompt: *"She turns around and smiles, then slowly walks out of the frame."*
 Download this image	 Prompt: *"The fridge door opens. A cute, chubby purple monster comes out of it."*

Remix completed videos

Remix lets you **take an existing video and make targeted adjustments** without regenerating everything from scratch. Provide the `remix_video_id` of a completed job along with a new prompt that describes the change, and the system reuses the original's structure, continuity, and composition while applying the modification. This works best when you make a **single, well-defined change** because smaller, focused edits preserve more of the original fidelity and reduce the risk of introducing artifacts.

```
curl -X POST "https://api.openai.com/v1/videos/<previous_video_id>/remix" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -H "Content-Type: application/json" \
  -d '{
    "prompt": "Shift the color palette to teal, sand, and rust, with a warm backlight."
  }'
```

Remix is especially valuable for iteration because it lets you refine without discarding what already works. By constraining each remix to one clear adjustment, you keep the visual style, subject consistency, and camera framing stable, while still exploring variations in mood, palette, or staging. This makes it far easier to build polished sequences through small, reliable steps.

ORIGINAL VIDEO	REMIX GENERATED VIDEO
	 Prompt: *"Change the color of the monster to orange."*
	

Prompt: *“A second monster comes out right after.”*

Maintain your library

Use `GET /videos` to enumerate your videos. The endpoint supports optional query parameters for pagination and sorting.

```
# default
curl "https://api.openai.com/v1/videos" \
  -H "Authorization: Bearer $OPENAI_API_KEY" | jq .
```

```
# with params
curl "https://api.openai.com/v1/videos?limit=20&after=video_123&order=asc" \
  -H "Authorization: Bearer $OPENAI_API_KEY" | jq .
```

Use `DELETE /videos/{video_id}` to remove videos you no longer need from OpenAI’s storage.

```
curl -X DELETE "https://api.openai.com/v1/videos/[REPLACE_WITH_YOUR_VIDEO_ID]" \
  -H "Authorization: Bearer $OPENAI_API_KEY" | jq .
```

Was this page useful?