Send User-Uploaded Images to ChatGPT with the OpenAI API

Generated from the previous response in this Codex thread. This guide shows how to change a Python app backend so the Run button sends an uploaded image plus a command to the OpenAI API, fetches the model result, and renders it back to the user.

Use the OpenAI **Responses API** for this. Your backend receives the uploaded image, converts it to an API-safe format, sends it with a text command or prompt, receives the model output, then returns that output to your UI. OpenAI supports text/image inputs with text output through the Responses API.

Official docs: Quickstart, Responses API, Image inputs.

1. Install the SDK

```
pip install openai python-dotenv pillow
```

If you use Flask, FastAPI, Streamlit, or another framework, keep your existing framework and add the OpenAI SDK to your backend environment.

2. Create an API Key

Create an API key in the OpenAI dashboard, then store it on the backend only:

```
export OPENAI_API_KEY="your_api_key_here"
```

Do not expose this key in browser JavaScript. API keys should be loaded from environment variables or server-side secret storage.

3. Decide How to Send the Image

For uploaded user images, the simplest option is usually a Base64 data URL. OpenAI supports image input as:

• A public image URL

• A Base64 data URL

• An OpenAI file_id from the Files API

For normal uploads from your app, use Base64. For larger or reused images, upload once with the Files API and pass the file_id.

4. Create a Backend Helper

```
# openai_image_job.py
import base64
import mimetypes
from openai import OpenAI

client = OpenAI()


def image_to_data_url(image_path: str) -> str:
```

```
    mime_type, _ = mimetypes.guess_type(image_path)

    if mime_type not in {"image/png", "image/jpeg", "image/webp", "image/gif"}:
        raise ValueError("Unsupported image type. Use PNG, JPEG, WEBP, or non-animated GIF.")

    with open(image_path, "rb") as image_file:
        encoded = base64.b64encode(image_file.read()).decode("utf-8")

    return f"data:{mime_type};base64,{encoded}"


def run_image_command(image_path: str, command: str) -> str:
    image_data_url = image_to_data_url(image_path)

    response = client.responses.create(
        model="gpt-4.1-mini",
        input=[
            {
                "role": "user",
                "content": [
                    {
                        "type": "input_text",
                        "text": command,
                    },
                    {
                        "type": "input_image",
                        "image_url": image_data_url,
                    },
                ],
            }
        ],
    )

    return response.output_text
```

You can replace gpt-4.1-mini with another vision-capable model available to your account. Use a stronger model for harder reasoning; use a mini/nano model for speed and cost.

5. Connect It to Your Run Button Backend

For Flask:

```
from flask import Flask, request, jsonify
from werkzeug.utils import secure_filename
import os
from openai_image_job import run_image_command

app = Flask(__name__)

UPLOAD_DIR = "uploads"
os.makedirs(UPLOAD_DIR, exist_ok=True)


@app.post("/run")
def run():
    image = request.files.get("image")
    command = request.form.get("command", "Analyze this image and return the result.")

    if image is None:
        return jsonify({"error": "No image uploaded"}), 400

    filename = secure_filename(image.filename)
    image_path = os.path.join(UPLOAD_DIR, filename)
```

```
    image.save(image_path)

    try:
        result = run_image_command(image_path, command)
        return jsonify({"result": result})
    except Exception as e:
        return jsonify({"error": str(e)}), 500
```

Frontend button flow:

```
async function runJob() {
  const formData = new FormData();
  formData.append("image", imageInput.files[0]);
  formData.append("command", commandInput.value);

  const res = await fetch("/run", {
    method: "POST",
    body: formData,
  });

  const data = await res.json();
  outputElement.textContent = data.result || data.error;
}
```

6. If You Use Streamlit Instead

```
import tempfile
import streamlit as st
from openai_image_job import run_image_command

st.title("Image Analyzer")

uploaded_file = st.file_uploader("Upload image", type=["png", "jpg", "jpeg", "webp", "gif"])
command = st.text_area("Command", "Describe this image in detail.")

if st.button("Run") and uploaded_file:
    suffix = "." + uploaded_file.name.split(".")[-1]

    with tempfile.NamedTemporaryFile(delete=False, suffix=suffix) as tmp:
        tmp.write(uploaded_file.read())
        image_path = tmp.name

    with st.spinner("Running..."):
        result = run_image_command(image_path, command)

    st.write(result)
```

7. Add Basic Safety and Reliability

Validate file type and size before sending the image. OpenAI image inputs support PNG, JPEG, WEBP, and non-animated GIF. Images are billed as tokens, so resizing very large images can reduce cost.

```
MAX_FILE_SIZE_MB = 10
ALLOWED_MIME_TYPES = {"image/png", "image/jpeg", "image/webp", "image/gif"}
```

Recommended production checks: loading state while the API call runs; try/except around OpenAI requests; friendly error messages; timeout and retry behavior; server-side logging; avoid storing uploads permanently unless needed.

8. If You Need JSON Output

Ask the model for JSON, then parse it:

```
command = """
Analyze the image and return JSON with:
{
  "summary": "...",
  "objects": ["..."],
  "confidence": "low|medium|high"
}
Return only valid JSON.
"""
```

For stricter production apps, use OpenAI Structured Outputs.

9. If the Output Should Be a New Image

The code above is for image input to text or JSON output. If you want image input to generated or edited image output, use OpenAI image generation instead. OpenAI image generation can accept input images by URL, Base64 data URL, or file ID.

Summary

Your Run button should call your Python backend. Your backend should call client.responses.create(...) with both input_text and input_image, then return response.output_text to the frontend.