



# Gen AI OpenAI and Gemini SDK

By PK



# OpenAI ChatGPT API

- We need to install the openai module to work with openai api in python
  - Pip install openai
- To check whether you have openai SDK or not
  - Pip show openai
- Get API key from <https://platform.openai.com/docs/overview>

```
from openai import OpenAI
client = OpenAI(api_key="YOUR_OPENAI_API_KEY")

response = client.chat.completions.create(
    model="gpt-3.5-turbo",
    messages=[
        {"role": "system", "content": "You are a helpful
assistant."},
        {"role": "user", "content": "Hello, how are you?"}
    ],
    temperature=0.7,
    max_tokens=150
)
print(response.choices[0].message.content)
```

```
curl https://api.openai.com/v1/responses \
-H "Content-Type: application/json" \
-H "Authorization: Bearer sk-proj-yd-belh2fVXr2YED2bJi" \
-d '{
    "model": "gpt-5-nano",
    "input": "write a haiku about ai",
    "store": true
}'
```



# ChatGPT Response json

```
{  
    "id": "chatcmpl-123",  
    "object": "chat.completion",  
    "created": 1699999999,  
    "model": "gpt-3.5-turbo",  
    "usage": {  
        "prompt_tokens": 25,  
        "completion_tokens": 45,  
        "total_tokens": 70  
    },  
    "choices": [  
        {  
            "message": {  
                "role": "assistant",  
                "content": "Hello! How can I assist you today?"  
            },  
            "finish_reason": "stop",  
            "index": 0  
        }  
    ]  
}
```

Field	Meaning
id	Unique ID for this chat completion.
object	Type of object, e.g., "chat.completion".
created	Unix timestamp of creation.
model	Model used for this response.
usage	Token usage info: prompt tokens, completion tokens, total tokens.
choices	A list of possible outputs (usually 1).
choices[i].message	Contains the actual content and role of the message (assistant).
choices[i].finish_reason	Why the model stopped: "stop", "length", "function_call", etc.

```
# To access from python  
reply = response.choices[0].message["content"]  
print(reply)
```



# ChatGPT chat template

```
{  
  "model": "gpt-3.5-turbo",  
  "messages": [  
    {"role": "system", "content": "You are a friendly assistant that helps with homework."},  
    {"role": "user", "content": "Can you explain photosynthesis?"}  
,  
  temperature=0.7  
}
```

Role	Purpose
<b>system</b>	Sets behavior, style, or rules for the model.
<b>user</b>	Represents the human asking a question or giving input.
<b>assistant</b>	(Optional) Represents the model's prior responses — useful for continuing multi-turn conversations.



# ChatGPT API

API Name	Endpoint	Primary Use Case	Key Features	Status/Recommendation
Chat Completions	/v1/chat/completions	Basic Chatbots & simple, stateless interactions.	Standard API for text/chat, widely adopted (Industry Standard), stateless (developer manages conversation history).	Recommended for basic, turn-based chat. Still actively supported.
Responses	/v1/responses	Advanced Agentic Workflows (Tool-use, complex reasoning).	Stateful (model/API handles conversation history), Built-in tools (Web Search, File Search, Code Interpreter), Persistent Reasoning (improves model intelligence), Structured Loop.	Recommended for new, complex agentic apps. Expected to become the default.
Completions	(Older Endpoint)	Legacy, freeform prompt style.	Simple text-in, text-out structure.	Deprecated/Legacy. Use Chat Completions or Responses instead.



# ChatGPT responses API

OpenAI is also rolling out a newer API called the **Responses API** (/v1/responses) which is designed for more advanced “agentic” use-cases (tool calling, reasoning state, etc)

```
response = client.responses.create(  
    model="gpt-3.5-turbo",  
    temperature=0.8,  
    input=[  
        {  
            "role": "system",  
            "content": "You are a friendly bedtime storyteller for children."  
        },  
        {  
            "role": "user",  
            "content": "Tell me a short bedtime story about a unicorn."  
        }  
    ]  
)  
  
print(response.output_text)
```



# ChatGPT responses API

the **Responses API** supports all these modalities, and you can mix them in a **single input array**.  
image\_url,image\_bytes,audio\_url,audio\_bytes,video\_uel and video\_bytes

```
with open("cat.png", "rb") as f:  
    image_bytes = f.read()  
with open("speech.mp3", "rb") as f:  
    audio_bytes = f.read()  
with open("video.mp4", "rb") as f:  
    video_bytes = f.read()  
  
response = client.responses.create(  
    model="gpt-4o-mini",  
    input=[  
        {"role": "system", "content": "You are a multimodal assistant."},  
        {"role": "user", "content": "Please summarize the contents of all these media."},  
        {"role": "user", "image_bytes": image_bytes},  
        {"role": "user", "audio_bytes": audio_bytes},  
        {"role": "user", "video_bytes": video_bytes},  
        # optional: URLs instead of bytes  
        {"role": "user", "image_url": "https://example.com/dog.jpg"},  
        {"role": "user", "audio_url": "https://example.com/speech.mp3"},  
        {"role": "user", "video_url": "https://example.com/video.mp4"}  
)  
  
print(response.output_text)
```



# Gemini via OpenAI-compatible interface

```
from openai import OpenAI

client = OpenAI(
    api_key="YOUR_GEMINI_API_KEY",
    base_url="https://generativelanguage.googleapis.com/v1beta/openai/"
)

response = client.chat.completions.create(
    model="gemini-2.0-flash",
    messages=[
        {"role": "system", "content": "You are a helpful assistant."},
        {"role": "user", "content": "Explain how a computer works."}
    ]
)
print(response.choices[0].message.content)
```



# Gemini via google-genai

## The New SDK: google-genai

This is the **modern** way. It was built to simplify the experience and unify Google's different AI platforms.

- **Install:** pip install google-genai
- **Import:** from google import genai

## The Old SDK: google-generativeai

This is the **legacy** version. You will see it in 90% of old YouTube tutorials and blog posts. It is officially on a "limited maintenance" path.

- **Install:** pip install google-generativeai
- **Import:** import google.generativeai as genai

Feature	New SDK (google-genai)	Old SDK (google-generativeai)
Status	Recommended (Current)	Deprecated / Legacy
Installation	pip install google-genai	pip install google-generativeai
Import style	from google import genai	import google.generativeai as genai
Architecture	Client-based (client.models...)	Object-based (model.generate...)
Gemini 2.0+	Native & optimized support.	Limited/Compatibility mode.
Multi-cloud	Works for AI Studio & Vertex AI.	Primarily for AI Studio.



# Gemini via google-genai

```
from google import genai
```

```
client = genai.Client(api_key="YOUR_KEY")
```

```
response = client.models.generate_content(  
    model="gemini-2.5-flash",  
    contents=[
```

```
        {  
            "role": "system",  
            "parts": [  
                {"text": "You are a helpful assistant who answers politely."}]
```

```
,  
        {
```

```
            "role": "user",  
            "parts": [  
                {"text": "Explain gravity in simple words."}]
```

```
        }  
    ]
```

```
)  
  
print(response.text)
```

## Multi-Modal

```
response = client.models.generate_content(  
    model="gemini-2.5-flash",  
    contents=[  
        {  
            "role": "user",  
            "parts": [  
                {"text": "Describe this image."},  
                {  
                    "inline_data": {  
                        "mime_type": "image/png",  
                        "data": open("cat.png", "rb").read()}}]}  
    ]  
)  
  
print(response.text)
```



**Thank you**