OpenAI Platform

Copy page

# Migrate to the Responses API

The Responses API is our new API primitive, an evolution of Chat Completions which brings added simplicity and powerful agentic primitives to your integrations.

**While Chat Completions remains supported, Responses is recommended for all new projects.**

## About the Responses API

The Responses API is a unified interface for building powerful, agent-like applications. It contains:

- Built-in tools like web search, file search , computer use, code interpreter, and remote MCPs.
- Seamless multi-turn interactions that allow you to pass previous responses for higher accuracy reasoning results.
- Native multimodal support for text and images.

## Responses benefits

The Responses API contains several benefits over Chat Completions:

- **Better performance**: Using reasoning models, like GPT-5, with Responses will result in better model intelligence when compared to Chat Completions. Our internal evals reveal a 3% improvement in SWE-bench with same prompt and setup.
- **Agentic by default**: The Responses API is an agentic loop, allowing the model to call multiple tools, like `web_search` , `image_generation` , `file_search` , `code_interpreter` , remote MCP servers, as well as your own custom functions, within the span of one API request.
- **Lower costs**: Results in lower costs due to improved cache utilization (40% to 80% improvement when compared to Chat Completions in internal tests).
- **Stateful context**: Use `store: true` to maintain state from turn to turn, preserving reasoning and tool context from turn-to-turn.
- **Flexible inputs**: Pass a string with input or a list of messages; use instructions for system-level guidance.

**Encrypted reasoning**: Opt-out of statefulness while still benefiting from advanced reasoning.

**Future-proof**: Future-proofed for upcoming models.

| CAPABILITIES | CHAT COMPLETIONS API | RESPONSES API |
| --- | --- | --- |
| Text generation | ✓ | ✓ |
| Audio | ✓ | Coming soon |
| Vision | ✓ | ✓ |
| Structured Outputs | ✓ | ✓ |
| Function calling | ✓ | ✓ |
| Web search | ✓ | ✓ |
| File search | ✗ | ✓ |
| Computer use | ✗ | ✓ |
| Code interpreter | ✗ | ✓ |
| MCP | ✗ | ✓ |
| Image generation | ✗ | ✓ |
| Reasoning summaries | ✗ | ✓ |

## Examples

See how the Responses API compares to the Chat Completions API in specific scenarios.

### Messages vs. Items

Both APIs make it easy to generate output from our models. The input to, and result of, a call to Chat completions is an array of *Messages*, while the Responses API uses *Items*. An Item is a union of many types, representing the range of possibilities of model actions. A `message` is a type of Item, as is a `function_call` or `function_call_output`. Unlike a Chat Completions Message, where many concerns are glued together into one object, Items are distinct from one another and better represent the basic unit of model context.

Additionally, Chat Completions can return multiple parallel generations as `choices`, using the `n` param. In Responses, we've removed this param, leaving only one generation.

Chat Completions API

```python
from openai import OpenAI
client = OpenAI()

completion = client.chat.completions.create(
  model="gpt-5",
  messages=[
      {
          "role": "user",
          "content": "Write a one-sentence bedtime story about a unicorn."
      }
  ]
)

print(completion.choices[0].message.content)
```

Responses API

```python
from openai import OpenAI
client = OpenAI()

response = client.responses.create(
  model="gpt-5",
  input="Write a one-sentence bedtime story about a unicorn."
)

print(response.output_text)
```

When you get a response back from the Responses API, the fields differ slightly. Instead of a `message`, you receive a typed `response` object with its own `id`. Responses are stored by default. Chat completions are stored by default for new accounts. To disable storage when using either API, set `store: false`.

The objects you recieve back from these APIs will differ slightly. In Chat Completions, you receive an array of `choices`, each containing a `message`. In Responses, you receive an array of Items labled `output`.

Chat Completions API

```
{
  "id": "chatcmpl-C9EDpkjH60VPPIB86j2zIhiR8kWiC",
```

```
  "object": "chat.completion",
  "created": 1756315657,
  "model": "gpt-5-2025-08-07",
  "choices": [
    {
      "index": 0,
      "message": {
        "role": "assistant",
        "content": "Under a blanket of starlight, a sleepy unicorn tiptoed thro
        "refusal": null,
        "annotations": []
      },
      "finish_reason": "stop"
    }
  ],
  ...
}
```

Responses API

```
{
  "id": "resp_68af4030592c81938ec0a5fbab4a3e9f05438e46b5f69a3b",
  "object": "response",
  "created_at": 1756315696,
  "model": "gpt-5-2025-08-07",
  "output": [
    {
      "id": "rs_68af4030baa48193b0b43b4c2a176a1a05438e46b5f69a3b",
      "type": "reasoning",
      "content": [],
      "summary": []
    },
    {
      "id": "msg_68af40337e58819392e935fb404414d005438e46b5f69a3b",
      "type": "message",
      "status": "completed",
      "content": [
        {
          "type": "output_text",
          "annotations": [],
          "logprobs": [],
          "text": "Under a quilt of moonlight, a drowsy unicorn wandered throug
        }
      ],
      "role": "assistant"
    }
  ],
```

```
  ...
}
```

### Additional differences

Responses are stored by default. Chat completions are stored by default for new accounts. To disable storage in either API, set `store: false`.

Reasoning models have a richer experience in the Responses API with improved tool usage.

Structured Outputs API shape is different. Instead of `response_format`, use `text.format` in Responses. Learn more in the Structured Outputs guide.

The function-calling API shape is different, both for the function config on the request, and function calls sent back in the response. See the full difference in the function calling guide.

The Responses SDK has an `output_text` helper, which the Chat Completions SDK does not have.

In Chat Completions, conversation state must be managed manually. The Responses API has compatibility with the Conversations API for persistent conversations, or the ability to pass a `previous_response_id` to easily chain Responses together.

## Migrating from Chat Completions

### 1. Update generation endpoints

Start by updating your generation endpoints from `post /v1/chat/completions` to `post /v1/responses`.

If you are not using functions or multimodal inputs, then you're done! Simple message inputs are compatible from one API to the other:

Web search tool javascript

```javascript
const context = [
  { role: 'system', content: 'You are a helpful assistant.' },
  { role: 'user', content: 'Hello!' }
];

const completion = await client.chat.completions.create({
  model: 'gpt-5',
  messages: messages
});
```

```
10
11 const response = await client.responses.create({
12   model: "gpt-5",
13   input: context
14 });
```

Chat Completions | Responses

With Chat Completions, you need to create an array of messages that specify different roles and content for each role.

Generate text from a model — javascript

```
1  import OpenAI from 'openai';
2  const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
3
4  const completion = await client.chat.completions.create({
5    model: 'gpt-5',
6    messages: [
7      { 'role': 'system', 'content': 'You are a helpful assistant.' },
8      { 'role': 'user', 'content': 'Hello!' }
9    ]
10 });
11 console.log(completion.choices[0].message.content);
```

## 2. Update item definitions

Chat Completions | Responses

With Chat Completions, you need to create an array of messages that specify different roles and content for each role.

Generate text from a model — javascript

```
1  import OpenAI from 'openai';
2  const client = new OpenAI({ apiKey: process.env.OPENAI_API_KEY });
3
4  const completion = await client.chat.completions.create({
5    model: 'gpt-5',
6    messages: [
7      { 'role': 'system', 'content': 'You are a helpful assistant.' },
8      { 'role': 'user', 'content': 'Hello!' }
```

```
    ]
});
console.log(completion.choices[0].message.content);
```

## 3. Update multi-turn conversations

If you have multi-turn conversations in your application, update your context logic.

Chat Completions | Responses

In Chat Completions, you have to store and manage context yourself.

Multi-turn conversation — javascript

```javascript
let messages = [
    { 'role': 'system', 'content': 'You are a helpful assistant.' },
    { 'role': 'user', 'content': 'What is the capital of France?' }
  ];
const res1 = await client.chat.completions.create({
  model: 'gpt-5',
  messages
});

messages = messages.concat([res1.choices[0].message]);
messages.push({ 'role': 'user', 'content': 'And its population?' });

const res2 = await client.chat.completions.create({
  model: 'gpt-5',
  messages
});
```

## 4. Decide when to use statefulness

Some organizations—such as those with Zero Data Retention (ZDR) requirements—cannot use the Responses API in a stateful way due to compliance or data retention policies. To support these cases, OpenAI offers encrypted reasoning items, allowing you to keep your workflow stateless while still benefiting from reasoning items.

To disable statefulness, but still take advantage of reasoning:

- set `store: false` in the store field
- add `["reasoning.encrypted_content"]` to the include field

The API will then return an encrypted version of the reasoning tokens, which you can pass back in future requests just like regular reasoning items. For ZDR organizations, OpenAI enforces store=false automatically. When a request includes encrypted_content, it is decrypted in-memory (never written to disk), used for generating the next response, and then securely discarded. Any new reasoning tokens are immediately encrypted and returned to you, ensuring no intermediate state is ever persisted.

## 5. Update function definitions

There are two minor, but notable, differences in how functions are defined between Chat Completions and Responses.

1. In Chat Completions, functions are defined using externally tagged polymorphism, whereas in Responses, they are internally-tagged.
2. In Chat Completions, functions are non-strict by default, whereas in the Responses API, functions *are* strict by default.

The Responses API function example on the right is functionally equivalent to the Chat Completions example on the left.

Chat Completions API

```
{
  "type": "function",
  "function": {
    "name": "get_weather",
    "description": "Determine weather in my location",
    "strict": true,
    "parameters": {
      "type": "object",
      "properties": {
        "location": {
          "type": "string",
        },
      },
      "additionalProperties": false,
      "required": [
        "location",
        "unit"
      ]
    }
  }
}
```

Responses API

```
{
  "type": "function",
  "name": "get_weather",
  "description": "Determine weather in my location",
  "parameters": {
    "type": "object",
    "properties": {
      "location": {
        "type": "string",
      },
    },
    "additionalProperties": false,
    "required": [
      "location",
      "unit"
    ]
  }
}
```

### Follow function-calling best practices

In Responses, tool calls and their outputs are two distinct types of Items that are correlated using a `call_id`. See the tool calling docs for more detail on how function calling works in Responses.

## 6. Update Structured Outputs definition

In the Responses API, defining structured outputs have moved from `response_format` to `text.format`:

Chat Completions | Responses

Structured Outputs — javascript

```javascript
const completion = await openai.chat.completions.create({
  model: "gpt-5",
  messages: [
    {
      "role": "user",
      "content": "Jane, 54 years old",
    }
  ],
  response_format: {
    type: "json_schema",
    json_schema: {
      name: "person",
      strict: true,
      schema: {
        type: "object",
        properties: {
          name: {
            type: "string",
            minLength: 1
          },
          age: {
            type: "number",
            minimum: 0,
            maximum: 130
          }
        },
        required: [
          name,
          age
        ],
```

```
        additionalProperties: false
      }
    }
  },
  verbosity: "medium",
  reasoning_effort: "medium"
});
```

## 7. Upgrade to native tools

If your application has use cases that would benefit from OpenAI's native tools, you can update your tool calls to use OpenAI's tools out of the box.

Chat Completions | Responses

With Chat Completions, you cannot use OpenAI's tools natively and have to write your own.

Web search tool — javascript

```javascript
async function web_search(query) {
    const fetch = (await import('node-fetch')).default;
    const res = await fetch(`https://api.example.com/search?q=${query}`);
    const data = await res.json();
    return data.results;
}

const completion = await client.chat.completions.create({
  model: 'gpt-5',
  messages: [
    { role: 'system', content: 'You are a helpful assistant.' },
    { role: 'user', content: 'Who is the current president of France?' }
  ],
  functions: [
    {
      name: 'web_search',
      description: 'Search the web for information',
      parameters: {
        type: 'object',
        properties: { query: { type: 'string' } },
        required: ['query']
      }
    }
  ]
});
```

## Incremental migration

The Responses API is a superset of the Chat Completions API. The Chat Completions API will also continue to be supported. As such, you can incrementally adopt the Responses API if desired. You can migrate user flows who would benefit from improved reasoning models to the Responses API while keeping other flows on the Chat Completions API until you're ready for a full migration.

As a best practice, we encourage all users to migrate to the Responses API to take advantage of the latest features and improvements from OpenAI.

## Assistants API

Based on developer feedback from the Assistants API beta, we've incorporated key improvements into the Responses API to make it more flexible, faster, and easier to use. The Responses API represents the future direction for building agents on OpenAI.

We now have Assistant-like and Thread-like objects in the Responses API. Learn more in the migration guide. As of August 26th, 2025, we're deprecating the Assistants API, with a sunset date of August 26, 2026.