OpenAI Platform

Using GPT-5.1

Copy page

Learn best practices, features, and migration guidance for GPT-5.1 and the GPT-5 model family.

GPT-5.1 is the newest flagship model, part of the GPT-5 model family. Our most intelligent model yet, GPT-5.1 has similar training for:

- Code generation, bug fixing, and refactoring
- Instruction following
- Long context and tool calling

Unlike the previous GPT-5 model, GPT-5.1 has a new `none` reasoning setting for faster responses, increased steerability in model output, and new tools for coding use cases.

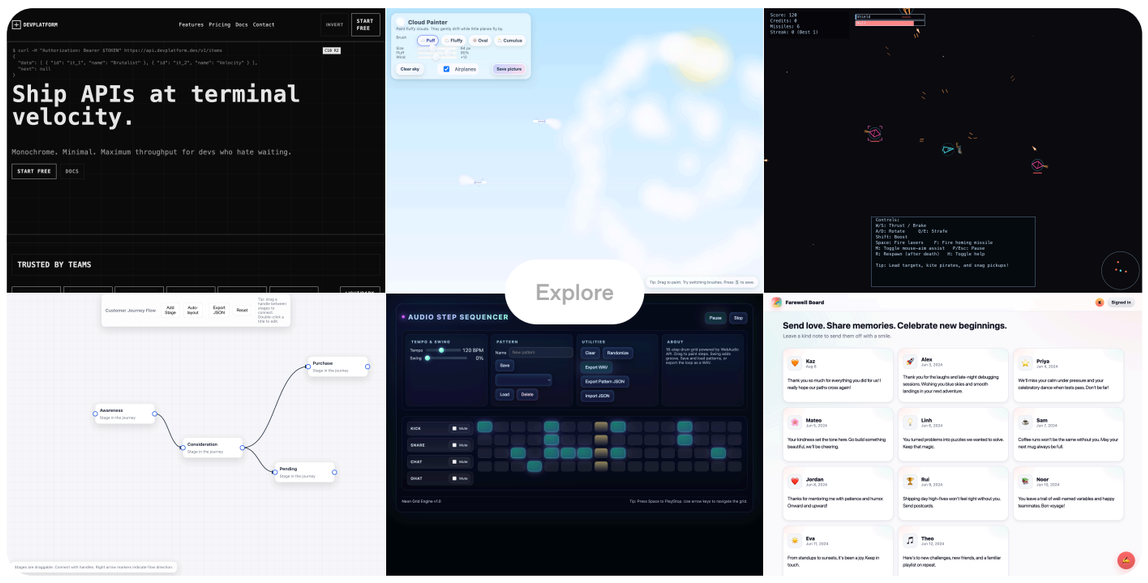
This guide covers key features of the GPT-5 model family and how to get the most out of GPT-5.1.

Overview
Quickstart
Meet the models
New features in GPT-5.1
Lower reasoning effort
Verbosity
New tool types
Custom tools
Allowed tools
Preambles
Migration guidance
Prompting guidance
Further reading
FAQ

Prompting guide
GPT-5 best practices

Explore coding examples

Click through a few demo applications generated entirely with a single GPT-5 prompt, without writing any code by hand. Note that these examples use our previous flagship model, GPT-5.



Quickstart

Faster responses | Coding and agentic tasks

GPT-5.1 has a new reasoning mode: `none` for low-latency interactions. By default, GPT-5.1 reasoning is set to `none`.

This behavior will more closely (but not exactly!) match non-reasoning models like GPT-4.1. We expect GPT-5.1 to produce more intelligent responses than GPT-4.1, but when speed and maximum context length are paramount, you might consider using GPT-4.1 instead.

Fast, low latency response options — python

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

result = client.responses.create(
    model="gpt-5.1",
    input="Write a haiku about code.",
    reasoning={ "effort": "low" },
    text={ "verbosity": "low" },
)

print(result.output_text)
```

Meet the models

There are three main models in the GPT-5 series. In general, `gpt-5.1` is best for your most complex tasks that require broad world knowledge. It replaces the previous `gpt-5` model. The smaller mini and nano models trade off some general world knowledge for lower cost and lower latency. Small models will tend to perform better for more well defined tasks.

To help you pick the model that best fits your use case, consider these tradeoffs:

VARIANT	BEST FOR
`gpt-5.1`	Complex reasoning, broad world knowledge, and code-heavy or multi-step agentic tasks
`gpt-5-mini`	Cost-optimized reasoning and chat; balances speed, cost, and capability
`gpt-5-nano`	High-throughput tasks, especially simple instruction-following or classification
`gpt-5`	Previous flagship model, replaced by `gpt-5.1`

New features in GPT-5.1

Just like GPT-5, the new GPT-5.1 has API features like custom tools, parameters to control verbosity and reasoning, and an allowed tools list. What's new in 5.1 is a `none` setting for reasoning effort, an increased steerability, and two new tools for coding use cases.

This guide walks through some of the key features of the GPT-5 model family and how to get the most out of these models.

Lower reasoning effort

The `reasoning.effort` parameter controls how many reasoning tokens the model generates before producing a response. Earlier reasoning models like o3 supported only `low`, `medium`, and `high`: `low` favored speed and fewer tokens, while `high` favored more thorough reasoning.

With GPT-5.1, the lowest setting is now `none` to provide lower-latency interactions. This is the default setting in GPT-5.1. If you need more thinking, slowly increase to `medium` and experiment with results. Note this difference from GPT-5's `minimal` reasoning setting and `medium` default.

With reasoning effort set to `none`, prompting is important. To improve the model's reasoning quality, even with the default settings, encourage it to "think" or outline its steps before answering.

Minimal reasoning effort — python

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

response = client.responses.create(
    model="gpt-5.1",
    input="How much gold would it take to coat the Statue of Liberty in a 1mm la
    reasoning={
        "effort": "none"
    }
)

print(response)
```

Verbosity

Verbosity determines how many output tokens are generated. Lowering the number of tokens reduces overall latency. While the model's reasoning approach stays mostly the same, the model finds ways to answer more concisely—which can either improve or diminish answer quality, depending on your use case. Here are some scenarios for both ends of the verbosity spectrum:

> **High verbosity:** Use when you need the model to provide thorough explanations of documents or perform extensive code refactoring.
>
> **Low verbosity:** Best for situations where you want concise answers or simple code generation, such as SQL queries.

GPT-5 made this option configurable as one of `high`, `medium`, or `low`. Now with GPT-5.1, verbosity remains configurable and defaults to `medium`.

When generating code, `medium` and `high` verbosity levels yield longer, more structured code with inline explanations, while `low` verbosity produces shorter, more concise code with minimal commentary.

Control verbosity — python

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

response = client.responses.create(
    model="gpt-5",
    input="What is the answer to the ultimate question of life, the universe, a
    text={
        "verbosity": "low"
    }
)

print(response)
```

You can still steer verbosity through prompting after setting it to `low` in the API. The verbosity parameter defines a general token range at the system prompt level, but the actual output is flexible to both developer and user prompts within that range.

New tool types in GPT-5.1

GPT-5.1 has been post-trained on specific tools commonly used in coding use cases.

The apply patch tool

The `apply_patch` tool lets GPT-5.1 create, update, and delete files in your codebase using structured diffs. Instead of just suggesting edits, the model emits patch operations that your application applies and then reports back on, enabling iterative, multistep code editing workflows. Read the docs.

This `apply_patch` is a new tool type in GPT-5.1, so you don't have to write custom descriptions for the tool. Under the hood, this implementation uses a freeform function call rather than a JSON format. In testing, the named function decreased `apply_patch` failure rates by 35%.

Shell tool

We've added a shell tool that allows the model to interact with your local computer through a controlled command-line interface. The model proposes shell commands; your integration executes them and returns the outputs. This creates a simple plan-execute loop that lets models inspect the system, run utilities, and gather data until they finish the task. Read the docs.

The shell tool is invoked in the same way as `apply_patch`: include it as a tool of type `shell`.

Custom tools

When the GPT-5 model family launched, we introduced a new capability called custom tools, which lets models send any raw text as tool call input but still constrain outputs if desired. This tool behavior remains true in GPT-5.1.

Function calling guide
Learn about custom tools in the function calling guide.

Freeform inputs

Define your tool with `type: custom` to enable models to send plaintext inputs directly to your tools, rather than being limited to structured JSON. The model can send any raw text—code, SQL queries, shell commands, configuration files, or long-form prose—directly to your tool.

```
{
    "type": "custom",
    "name": "code_exec",
    "description": "Executes arbitrary python code",
}
```

Constraining outputs

GPT-5.1 supports context-free grammars (CFGs) for custom tools, letting you provide a Lark grammar to constrain outputs to a specific syntax or DSL. Attaching a CFG (e.g., a SQL or DSL grammar) ensures the assistant's text matches your grammar.

This enables precise, constrained tool calls or structured responses and lets you enforce strict syntactic or domain-specific formats directly in GPT-5.1's function calling, improving control and reliability for complex or constrained domains.

Best practices for custom tools

Write concise, explicit tool descriptions. The model chooses what to send based on your description; state clearly if you want it to always call the tool.

Validate outputs on the server side. Freeform strings are powerful but require safeguards against injection or unsafe commands.

Allowed tools

The `allowed_tools` parameter under `tool_choice` lets you pass N tool definitions but restrict the model to only M (< N) of them. List your full toolkit in `tools`, and then use an `allowed_tools` block to name the subset and specify a mode—either `auto` (the model may pick any of those) or `required` (the model must invoke one).

Function calling guide
Learn about the allowed tools option in the function calling guide.

By separating all possible tools from the subset that can be used *now*, you gain greater safety, predictability, and improved prompt caching. You also avoid brittle prompt engineering, such as hard-coded call order. GPT-5.1 dynamically invokes or requires specific functions mid-conversation while reducing the risk of unintended tool usage over long contexts.

	STANDARD TOOLS	ALLOWED TOOLS
Model's universe	All tools listed under **`"tools": […]`**	Only the subset under **`"tools": […]`** in **`tool_choice`**
Tool invocation	Model may or may not call any tool	Model restricted to (or required to call) chosen tools
Purpose	Declare available capabilities	Constrain which capabilities are actually used

```
"tool_choice": {
    "type": "allowed_tools",
    "mode": "auto",
    "tools": [
      { "type": "function", "name": "get_weather" },
      { "type": "function", "name": "search_docs" }
    ]
  }
}'
```

For a more detailed overview of all of these new features, see the accompanying cookbook.

Preambles

Preambles are brief, user-visible explanations that GPT-5.1 generates before invoking any tool or function, outlining its intent or plan (e.g., “why I'm calling this tool”). They appear after the chain-of-thought and before the actual tool call, providing transparency into the model's reasoning and enhancing debuggability, user confidence, and fine-grained steerability.

By letting GPT-5.1 “think out loud” before each tool call, preambles boost tool-calling accuracy (and overall task success) without bloating reasoning overhead. To enable preambles, add a system or developer instruction—for example: “Before you call a tool, explain why you are calling it.” GPT-5.1 prepends a concise rationale to each specified tool call. The model may also output multiple messages between tool calls, which can enhance the interaction experience—particularly for minimal reasoning or latency-sensitive use cases.

For more on using preambles, see the GPT-5.1 prompting cookbook.

Migration guidance

GPT-5.1 is our best model yet, and it works best with the Responses API, which supports for passing chain of thought (CoT) between turns. Read below to migrate from your current model or API.

Migrating from other models to GPT-5.1

We have seen significant success with customers switching from GPT-5 to GPT-5.1. While the model should be close to a drop-in replacement for GPT-5, there are a few key changes to call out. See the GPT-5.1 prompting guide for specific updates to make in your prompts to handle changes in persistence, output formatting and verbosity, coding use cases, and instruction following.

Across models in the GPT-5 family, we see improved intelligence because the Responses API can pass the previous turn's CoT to the model. This leads to fewer generated reasoning tokens, higher cache hit rates, and less latency. To learn more, see an in-depth guide on the benefits of responses.

When migrating to GPT-5.1 from an older OpenAI model, start by experimenting with reasoning levels and prompting strategies. Based on our testing, we recommend using our prompt optimizer —which automatically updates your prompts for GPT-5.1 based on our best practices—and following this model-specific guidance:

- **gpt-5**: `gpt-5.1` with default settings is meant to be a drop-in replacement.
- **o3**: `gpt-5.1` with `medium` or `high` reasoning is a great replacement. Start with `medium` reasoning with prompt tuning, then increasing to `high` if you aren't getting the results you want.
- **gpt-4.1**: `gpt-5` with `none` reasoning is a strong alternative. Start with `none` and tune your prompts; increase if you need better performance.
- **o4-mini or gpt-4.1-mini**: `gpt-5-mini` with prompt tuning is a great replacement.
- **gpt-4.1-nano**: `gpt-5-nano` with prompt tuning is a great replacement.

GPT-5.1 parameter compatibility

⚠️ **Important:** The following parameters are **not supported** when using GPT-5 models (e.g. `gpt-5.1`, `gpt-5`, `gpt-5-mini`, `gpt-5-nano`):

- `temperature`
- `top_p`
- `logprobs`

Requests that include these fields will raise an error.

Instead, use the following controls specific to the GPT-5 model family:

- **Reasoning depth:** `reasoning: { effort: "none" | "low" | "medium" | "high" }`
- **Output verbosity:** `text: { verbosity: "low" | "medium" | "high" }`
- **Output length:** `max_output_tokens`

Migrating from Chat Completions to Responses API

The biggest difference, and main reason to migrate from Chat Completions to the Responses API for GPT-5.1, is support for passing chain of thought (CoT) between turns. See a full comparison of the APIs.

Passing CoT exists only in the Responses API, and we've seen improved intelligence, fewer generated reasoning tokens, higher cache hit rates, and lower latency as a result of doing so. Most other parameters remain at parity, though the formatting is different. Here's how new parameters are handled differently between Chat Completions and the Responses API:

Reasoning effort

Responses API | Chat Completions

Generate response with minimal reasoning

```
curl --request POST \
--url https://api.openai.com/v1/responses \
--header "Authorization: Bearer $OPENAI_API_KEY" \
--header 'Content-type: application/json' \
--data '{
  "model": "gpt-5.1",
  "input": "How much gold would it take to coat the Statue of Liberty in a 1mm
  "reasoning": {
    "effort": "none"
  }
}'
```

Verbosity

Responses API | Chat Completions

Control verbosity

```
curl --request POST \
--url https://api.openai.com/v1/responses \
--header "Authorization: Bearer $OPENAI_API_KEY" \
--header 'Content-type: application/json' \
--data '{
  "model": "gpt-5.1",
  "input": "What is the answer to the ultimate question of life, the universe, a
  "text": {
    "verbosity": "low"
```

```
  }
}'
```

Custom tools

Responses API | Chat Completions

Custom tool call

```
curl --request POST --url https://api.openai.com/v1/responses --header "Authori
  "model": "gpt-5.1",
  "input": "Use the code_exec tool to calculate the area of a circle with radiu
  "tools": [
    {
      "type": "custom",
      "name": "code_exec",
      "description": "Executes arbitrary python code"
    }
  ]
}'
```

Prompting guidance

We specifically designed GPT-5.1 to excel at coding and agentic tasks. We also recommend iterating on prompts for GPT-5.1 using the prompt optimizer.

GPT-5.1 prompt optimizer
Craft the perfect prompt for GPT-5.1 in the dashboard

GPT-5.1 prompting guide
Learn full best practices for prompting GPT-5 models

Frontend prompting for GPT-5
See prompt samples specific to frontend development for GPT-5 family of models

GPT-5.1 is a reasoning model

Reasoning models like GPT-5.1 break problems down step by step, producing an internal chain of thought that encodes their reasoning. To maximize performance, pass these reasoning items back to the model: this avoids re-reasoning and keeps interactions closer to the model's training distribution. In multi-turn conversations, passing a `previous_response_id` automatically makes earlier reasoning items available. This is especially important when using tools—for example, when a function call requires an extra round trip. In these cases, either include them with `previous_response_id` or add them directly to `input`.

Learn more about reasoning models and how to get the most out of them in our reasoning guide.

Further reading

GPT-5.1 prompting guide

GPT-5 frontend guide

GPT-5 new features guide

Cookbook on reasoning models

Comparison of Responses API vs. Chat Completions

FAQ

1. **How are these models integrated into ChatGPT?**

   In ChatGPT, there are two models: GPT-5.1 Instant and GPT-5.1 Thinking. They offer reasoning and minimal-reasoning capabilities, with a routing layer that selects the best model based on the user's question. Users can also invoke reasoning directly through the ChatGPT UI.

2. **Will these models be supported in Codex?**

   Yes, `gpt-5.1` will be available in Codex and Codex CLI.

3. **How does GPT-5.1 compare to GPT-5-Codex?**

   GPT-5.1-Codex was specifically designed for use in Codex. Unlike GPT-5.1, which is a general-purpose model, we recommend using GPT-5.1-Codex only for agentic coding tasks in Codex or Codex-like environments, and GPT-5.1 for use cases in other domains. GPT-5.1-Codex is only available in the Responses API and supports `none`, `medium`, and `high` reasoning effort settings as well function calling, structured outputs, and the `web_search` tool.

4. **What is the deprecation plan for previous models?**

   Any model deprecations will be posted on our deprecations page. We'll send advanced notice of any model deprecations.