Reasoning models

Copy page

Explore advanced reasoning and problem-solving models.

Reasoning models, like OpenAI o1 and o3-mini, are new large language models trained with reinforcement learning to perform complex reasoning. Reasoning models think before they answer, producing a long internal chain of thought before responding to the user. Reasoning models excel in complex problem solving, coding, scientific reasoning, and multi-step planning for agentic workflows.

As with our GPT models, we provide both a smaller, faster model (`o3-mini`) that is less expensive per token, and a larger model (`o1`) that is somewhat slower and more expensive, but can often generate better responses for complex tasks, and generalize better across domains.

Quickstart

Reasoning models can be used through the chat completions endpoint as seen here.

Using a reasoning model in chat completions

javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const prompt = `
Write a bash script that takes a matrix represented as a string with
format '[1,2],[3,4],[5,6]' and prints the transpose in the same format.
`;

const completion = await openai.chat.completions.create({
  model: "o3-mini",
  reasoning_effort: "medium",
  messages: [
    {
      role: "user",
      content: prompt
    }
  ],
  store: true,
});

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

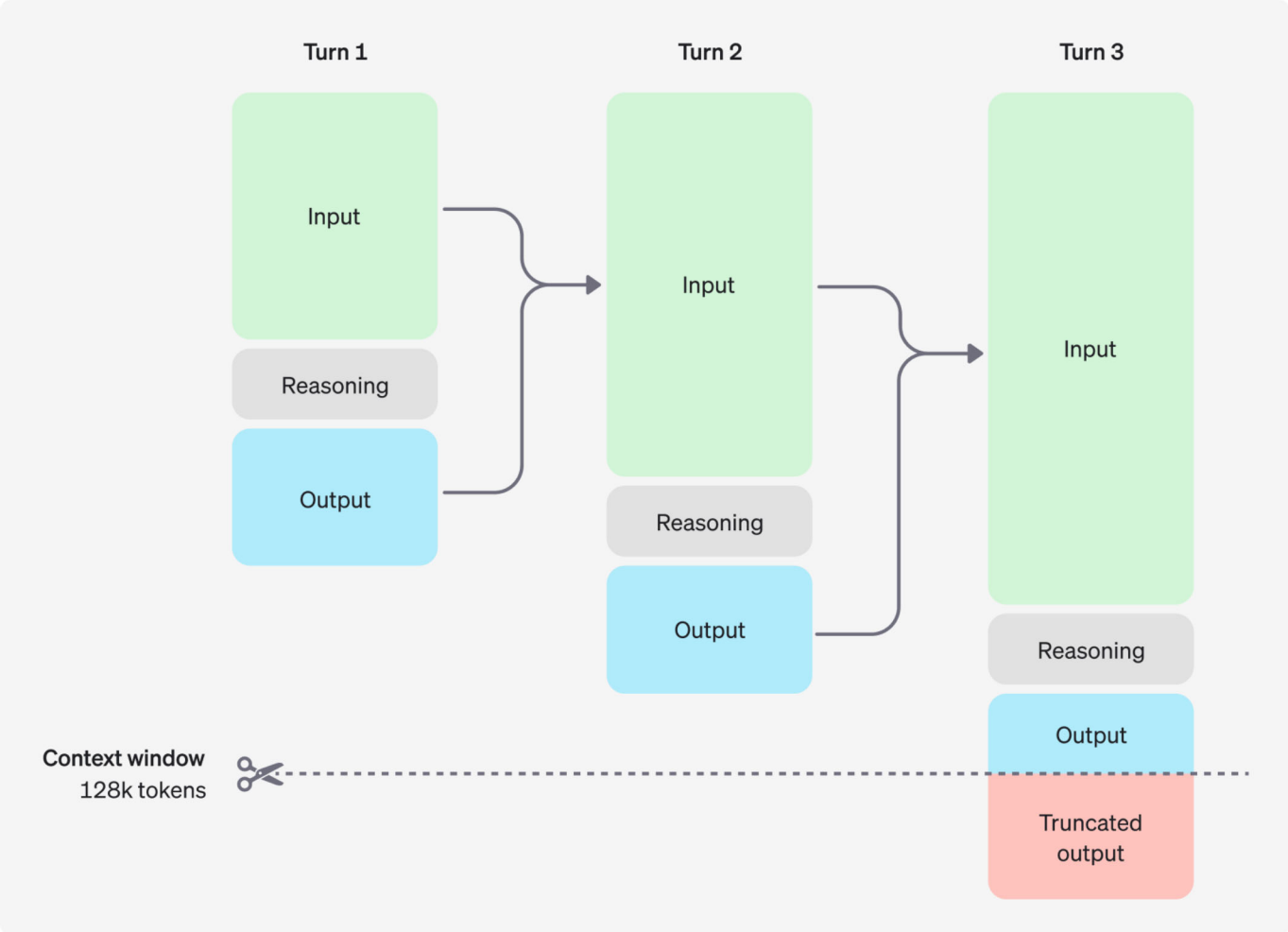
Reasoning effort

In the examples above, the `reasoning_effort` parameter (lovingly referred to as the "juice" during the development of these models) is used to give the model guidance on how many reasoning tokens it should generate before creating a response to the prompt. You can specify one of `low`, `medium`, or `high` for this parameter, where `low` will favor speed and economical token usage, and `high` will favor more complete reasoning at the cost of more tokens generated and slower responses. The default value is `medium`, which is a balance between speed and reasoning accuracy.

How reasoning works

Reasoning models introduce **reasoning tokens** in addition to input and output tokens. The models use these reasoning tokens to "think", breaking down their understanding of the prompt and considering multiple approaches to generating a response. After generating reasoning tokens, the model produces an answer as visible completion tokens, and discards the reasoning tokens from its context.

Here is an example of a multi-step conversation between a user and an assistant. Input and output tokens from each step are carried over, while reasoning tokens are discarded.



While reasoning tokens are not visible via the API, they still occupy space in the model's context window and are billed as output tokens.

Managing the context window

It's important to ensure there's enough space in the context window for reasoning tokens when creating completions. Depending on the problem's complexity, the models may generate anywhere from a few hundred to tens of thousands of reasoning tokens. The exact number of reasoning tokens used is visible in the usage object of the chat completion response object, under `completion_tokens_details`:

```
{
  "usage": {
    "prompt_tokens": 9,
    "completion_tokens": 12,
    "total_tokens": 21,
    "completion_tokens_details": {
      "reasoning_tokens": 0,
      "accepted_prediction_tokens": 0,
      "rejected_prediction_tokens": 0
    }
  }
}
```

Context window lengths are found on the model reference page.

Controlling costs

To manage costs with reasoning models, you can limit the total number of tokens the model generates (including both reasoning and completion tokens) by using the `max_completion_tokens` parameter.

In previous models, the `max_tokens` parameter controlled both the number of tokens generated and the number of tokens visible to the user, which were always equal. However, with reasoning models, the total tokens generated can exceed the number of visible tokens due to the internal reasoning tokens.

Because some applications might rely on `max_tokens` matching the number of tokens received from the API, we introduced `max_completion_tokens` to explicitly control the total number of tokens generated by the model, including both reasoning and visible completion tokens. This explicit opt-in ensures no existing applications break when using the new models. The `max_tokens` parameter continues to function as before for all previous models.

Allocating space for reasoning

If the generated tokens reach the context window limit or the `max_completion_tokens` value you've set, you'll receive a chat completion response with the `finish_reason` set to `length`. This might occur before any visible completion tokens are produced, meaning you could incur costs for input and reasoning tokens without receiving a visible response.

To prevent this, ensure there's sufficient space in the context window or adjust the `max_completion_tokens` value to a higher number. OpenAI recommends reserving at least

25,000 tokens for reasoning and outputs when you start experimenting with these models. As you become familiar with the number of reasoning tokens your prompts require, you can adjust this buffer accordingly.

Advice on prompting

There are some differences to consider when prompting a reasoning model versus prompting a GPT model. Generally speaking, reasoning models will provide better results on tasks with only high-level guidance. This differs somewhat from GPT models, which often benefit from very precise instructions.

> A reasoning model is like a senior co-worker - you can give them a goal to achieve, and trust them to work out the details.
>
> A GPT model is like a junior co-worker - they will perform best with explicit instructions to create a specific output.

For more information on best practices when using reasoning models, refer to this guide.

Prompt examples

Coding (refactoring) | Coding (planning) | STEM Research

OpenAI o-series models are able to implement complex algorithms and produce code. This prompt asks o1 to refactor a React component based on some specific criteria.

Refactor code — javascript

```javascript
import OpenAI from "openai";
const openai = new OpenAI();

const prompt = `
Instructions:
- Given the React component below, change it so that nonfiction books have red
  text.
- Return only the code in your reply
- Do not include any additional formatting, such as markdown code blocks
- For formatting, use four space tabs, and do not allow any lines of code to
  exceed 80 columns

const books = [
  { title: 'Dune', category: 'fiction', id: 1 },
  { title: 'Frankenstein', category: 'fiction', id: 2 },
  { title: 'Moneyball', category: 'nonfiction', id: 3 },
];

export default function BookList() {
  const listItems = books.map(book =>
    <li>
      {book.title}
```

```
      </li>
    );

    return (
      <ul>{listItems}</ul>
    );
  }
  `.trim();

  const completion = await openai.chat.completions.create({
    model: "o3-mini",
    messages: [
      {
        role: "user",
        content: prompt,
      },
    ],
    store: true,
  });

  console.log(completion.usage.completion_tokens_details);
```

Use case examples

Some examples of using reasoning models for real-world use cases can be found in the cookbook.

Using reasoning for data validation

Evaluate a synthetic medical data set for discrepancies.

Using reasoning for routine generation

Use help center articles to generate actions that an agent could perform.