M17b - Reasoning models

> **Reasoning Models - OpenAI**
>
> Quelle: Reasoning models - OpenAI API

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.

The new o1-pro model has unique features like making multiple model generation turns before generating a response. To support this and other advanced API features in the future, this model is currently only available in the Responses API.

1 Get started with reasoning

Reasoning models can be used through the Responses API as seen here.

Using a reasoning model in the Responses API

```
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 response = await openai.responses.create({
    model: "o3-mini",
    reasoning: { effort: "medium" },
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
});

console.log(response.output_text);
```

```
from openai import OpenAI

client = OpenAI()

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.
"""

response = client.responses.create(
    model="o3-mini",
    reasoning={"effort": "medium"},
    input=[
        {
            "role": "user",
            "content": prompt
        }
    ]
)

print(response.output_text)
```

```
curl https://api.openai.com/v1/responses \
  -H "Content-Type: application/json" \
  -H "Authorization: Bearer $OPENAI_API_KEY" \
  -d '{
    "model": "o3-mini",
    "reasoning": {"effort": "medium"},
    "input": [
      {
        "role": "user",
        "content": "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."
      }
    ]
  }'
```

Reasoning effort

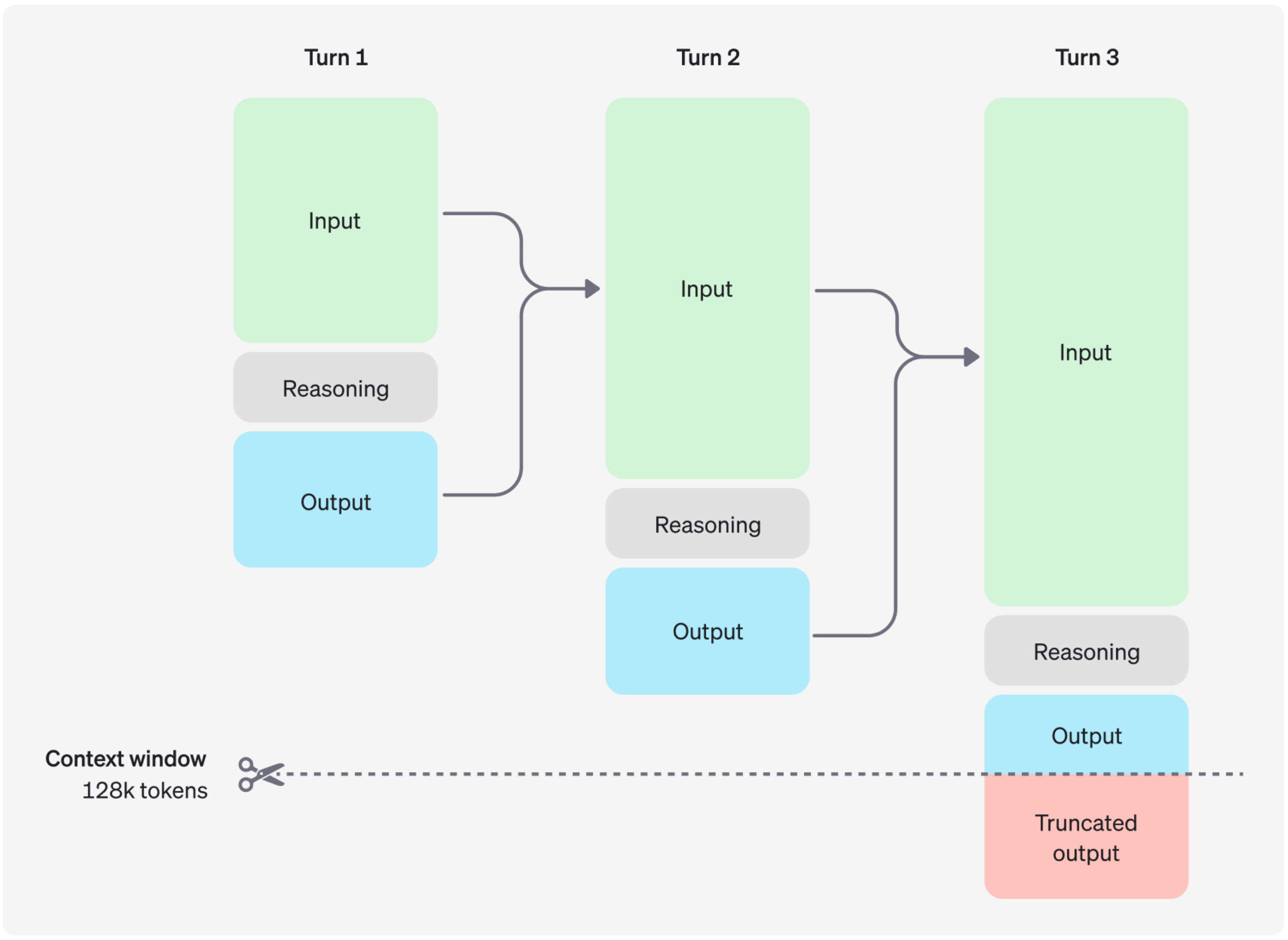
In the example above, the `reasoning.effort` parameter is used to give the model guidance on how many reasoning tokens it should generate before creating a response to the prompt.

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.

2 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 responses. 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 response object, under `output_tokens_details`:

```
{
    "usage": {
        "input_tokens": 75,
        "input_tokens_details": {
            "cached_tokens": 0
        },
        "output_tokens": 1186,
        "output_tokens_details": {
            "reasoning_tokens": 1024
        },
        "total_tokens": 1261
    }
}
```

Context window lengths are found on the model reference page, and will differ across model snapshots.

Controlling costs

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

Allocating space for reasoning

If the generated tokens reach the context window limit or the `max_output_tokens` value you've set, you'll receive a response with a `status` of `incomplete` and `incomplete_details` with `reason` set to `max_output_tokens`. This might occur before any visible output 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_output_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.

Handling incomplete responses

```
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 response = await openai.responses.create({
    model: "o3-mini",
    reasoning: { effort: "medium" },
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
    max_output_tokens: 300,
});

if (
    response.status === "incomplete" &&
    response.incomplete_details.reason === "max_output_tokens"
) {
    console.log("Ran out of tokens");
    if (response.output_text?.length > 0) {
        console.log("Partial output:", response.output_text);
    } else {
        console.log("Ran out of tokens during reasoning");
    }
}
```

```
from openai import OpenAI

client = OpenAI()

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.
"""

response = client.responses.create(
    model="o3-mini",
    reasoning={"effort": "medium"},
    input=[
        {
            "role": "user",
            "content": prompt
        }
    ],
    max_output_tokens=300,
)

if response.status == "incomplete" and response.incomplete_details.reason ==
"max_output_tokens":
    print("Ran out of tokens")
    if response.output_text:
        print("Partial output:", response.output_text)
    else:
        print("Ran out of tokens during reasoning")
```

3 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 coworker—they'll 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)

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

```
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 response = await openai.responses.create({
    model: "o3-mini",
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
});

console.log(response.output_text);
```

```
from openai import OpenAI

client = OpenAI()

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>
  );
}
"""

response = client.responses.create(
    model="o3-mini",
    input=[
        {
            "role": "user",
            "content": prompt,
        }
    ]
)

print(response.output_text)
```

Coding (planning)

OpenAI o-series models are also adept in creating multi-step plans. This example prompt asks o1 to create a filesystem structure for a full solution, along with Python code that implements the desired use case.

Plan and create a Python project

```
import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `
I want to build a Python app that takes user questions and looks
them up in a database where they are mapped to answers. If there
is close match, it retrieves the matched answer. If there isn't,
it asks the user to provide an answer and stores the
question/answer pair in the database. Make a plan for the directory
structure you'll need, then return each file in full. Only supply
your reasoning at the beginning and end, not throughout the code.
`.trim();

const response = await openai.responses.create({
    model: "o3-mini",
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
});

console.log(response.output_text);
```

```
from openai import OpenAI

client = OpenAI()

prompt = """
I want to build a Python app that takes user questions and looks
them up in a database where they are mapped to answers. If there
is close match, it retrieves the matched answer. If there isn't,
it asks the user to provide an answer and stores the
question/answer pair in the database. Make a plan for the directory
structure you'll need, then return each file in full. Only supply
your reasoning at the beginning and end, not throughout the code.
"""

response = client.responses.create(
    model="o3-mini",
    input=[
        {
            "role": "user",
            "content": prompt,
        }
    ]
)

print(response.output_text)
```

STEM Research (*Science, Technology, Engineering and Mathematics*)

OpenAI o-series models have shown excellent performance in STEM research. Prompts asking for support of basic research tasks should show strong results.

Ask questions related to basic scientific research

```javascript
import OpenAI from "openai";

const openai = new OpenAI();

const prompt = `
What are three compounds we should consider investigating to
advance research into new antibiotics? Why should we consider
them?
`;

const response = await openai.responses.create({
    model: "o3-mini",
    input: [
        {
            role: "user",
            content: prompt,
        },
    ],
});

console.log(response.output_text);
```

```python
from openai import OpenAI

client = OpenAI()

prompt = """
What are three compounds we should consider investigating to
advance research into new antibiotics? Why should we consider
them?
"""

response = client.responses.create(
    model="o3-mini",
    input=[
        {
            "role": "user",
            "content": prompt
        }
    ]
)

print(response.output_text)
```