Jun 25, 2025

Introduction to deep research in the OpenAI API

Glory Jain, Kevin Alwell (OpenAI)

Open in Github View as Markdown

Background

The Deep Research API enables you to automate complex research workflows that require reasoning, planning, and synthesis across real-world information. It is designed to take a high-level query and return a structured, citation-rich report by leveraging an agentic model capable of decomposing the task, performing web searches, and synthesizing results.

Unlike ChatGPT where this process is abstracted away, the API provides direct programmatic access. When you send a request, the model autonomously plans sub-questions, uses tools like web search and code execution, and produces a final structured response. This cookbook will provide a brief introduction to the Deep Research API and how to use it.

You can access Deep Research via the `responses` endpoint using the following models:

- `o3-deep-research-2025-06-26`: Optimized for in-depth synthesis and higher-quality output
- `o4-mini-deep-research-2025-06-26`: Lightweight and faster, ideal for latency-sensitive use cases

Setup

Install requirements

Install the latest version of the OpenAI Python SDK.

```
!pip install --upgrade openai
```

Authenticate

Import the OpenAI client and initialize with your API key.

```
from openai import OpenAI
OPENAI_API_KEY="" # YOUR OPENAI_API_KEY
#OPENAI_API_KEY = os.environ.get("OPENAI_API_KEY")
client = OpenAI(api_key=OPENAI_API_KEY)
```

Getting started

Let’s walk through an example of a Deep Research API call. Imagine we’re working at a healthcare financial services firm tasked with producing an in-depth report on the economic implications of recent medications used to treat type 2 diabetes and obesity—particularly semaglutide. Our goal is to synthesize clinical outcomes, cost-effectiveness, and regional pricing data into a structured, citation-backed analysis that could inform investment, payer strategy, or policy recommendations.

To get started, let's:

- Put our role in the system message, outlining what type of report we'd like to generate
- Set the summary paramter to "auto" for now for the best available summary. (If you'd like for your report to more detailed, you can set summary to detailed)
- Include the required tool web_search_preview and optionally add code_interpreter.
- Set the background parameter to True. Since a Deep Research task can take several minutes to execute, enabling background mode will allow you to run the request asynchronously without having to worry about timeouts or other connectivity issues.

```
system_message = """
You are a professional researcher preparing a structured, data-driven report on behalf of a

Do:
- Focus on data-rich insights: include specific figures, trends, statistics, and measurable
- When appropriate, summarize data in a way that could be turned into charts or tables, and
- Prioritize reliable, up-to-date sources: peer-reviewed research, health organizations (e.
- Include inline citations and return all source metadata.

Be analytical, avoid generalities, and ensure that each section supports data-backed reason
"""

user_query = "Research the economic impact of semaglutide on global healthcare systems."

response = client.responses.create(
  model="o3-deep-research",
  input=[
    {
      "role": "developer",
      "content": [
        {
          "type": "input_text",
          "text": system_message,
        }
      ]
    },
    {
      "role": "user",
      "content": [
        {
          "type": "input_text",
          "text": user_query,
        }
      ]
    }
  ],
  reasoning={
    "summary": "auto"
  },
  tools=[
    {
      "type": "web_search_preview"
    },
    {
      "type": "code_interpreter",
      "container": {
        "type": "auto",
        "file_ids": []
      }
    }
  ]
)
```

Parse the Response

The Deep Research API response includes a structured final answer along with inline citations, summaries of the reasoning steps, and source metadata.

Extract the Final Report Output

Here's the main text output of this report.

```
# Access the final report from the response object
print(response.output[-1].content[0].text)
```

Access Inline Citations and Metadata

Inline citations in the response text are annotated and linked to their corresponding source metadata. Each annotation contains:

- start_index and end_index: the character span in the text the citation refers to
- title: a brief title of the source
- url: the full source URL

This structure will allow you to build a citation list or bibliography, add clickable hyperlinks in downstream apps, and highlight & trace data-backed claims in your report.

```
annotations = response.output[-1].content[0].annotations
for i, citation in enumerate(annotations):
    print(f"Citation {i+1}:")
    print(f"  Title: {citation.title}")
    print(f"  URL: {citation.url}")
    print(f"  Location: chars {citation.start_index}–{citation.end_index}")
```

Inspect Intermediate Steps

The Deep Research API also exposes all intermediate steps taken by the agent, including reasoning steps, web search calls, and code executions. You can use these to debug, analyze, or visualize how the final answer was constructed. Each intermediate step is stored in `response.output`, and the `type` field indicates what kind it is.

Reasoning Step

These represent internal summaries or plans generated by the model as it reasons through sub-questions.

```
# Find the first reasoning step
reasoning = next(item for item in response.output if item.type == "reasoning")
for s in reasoning.summary:
    print(s.text)
```

Web Search Call

These show what search queries were executed and can help you trace what information the model retrieved.

```
# Find the first web search step
search = next(item for item in response.output if item.type == "web_search_call")
print("Query:", search.action["query"])
print("Status:", search.status)
```

Code Execution

If the model used the code interpreter (e.g. for parsing data or generating charts), those steps will appear as type "code_interpreter_call" or similar.

```
# Find a code execution step (if any)
code_step = next((item for item in response.output if item.type == "code_interpreter_call")
if code_step:
    print(code_step.input)
    print(code_step.output)
else:
    print("No code execution steps found.")
```

Model Context Protocol (MCP)

Suppose you would like to pull in your own internal documents as part of a Deep Research task. The Deep Research models and the Responses API both support MCP-based tools, so you can extend them to query your private knowledge stores or other 3rd party services.

In the example below, we configure an MCP tool that lets Deep Research fetch your organizations internal semaglutide studies on demand. The MCP server is a proxy for the OpenAI File Storage service that automagically vectorizes your uploaded files for performant retrieval.

If you would like to see *how* we built this simple MCP server, refer to this related cookbook.

```python
# system_message includes reference to internal file lookups for MCP.
system_message = """
You are a professional researcher preparing a structured, data-driven report on behalf of a

Do:
- Focus on data-rich insights: include specific figures, trends, statistics, and measurable
- When appropriate, summarize data in a way that could be turned into charts or tables, and
- Prioritize reliable, up-to-date sources: peer-reviewed research, health organizations (e.
- Include an internal file lookup tool to retrieve information from our own internal data s
- Include inline citations and return all source metadata.

Be analytical, avoid generalities, and ensure that each section supports data-backed reason
"""

user_query = "Research the economic impact of semaglutide on global healthcare systems."

response = client.responses.create(
  model="o3-deep-research-2025-06-26",
  input=[
    {
      "role": "developer",
      "content": [
        {
          "type": "input_text",
          "text": system_message,
        }
      ]
    },
    {
      "role": "user",
      "content": [
        {
          "type": "input_text",
          "text": user_query,
        }
      ]
    }
  ],
  reasoning={
    "summary": "auto"
  },
  tools=[
    {
      "type": "web_search_preview"
    },
    { # ADD MCP TOOL SUPPORT
      "type": "mcp",
      "server_label": "internal_file_lookup",
      "server_url": "https://<your_mcp_server>/sse/", # Update to the location of *your* MC
      "require_approval": "never"
    }
  ]
)
```

Reviewing your response

First 100 characters of your Research Report, followed by Citations and MCP tool calls.

```
# Grab the full report text once
report_text = response.output[-1].content[0].text

print("REPORT EXCERPT:")
print(report_text[:100])   # first 100 chars
print("--------")

annotations = response.output[-1].content[0].annotations
target_url = "https://platform.openai.com/storage/files"

for citation in annotations:
    if citation.url.startswith(target_url):
        start, end = citation.start_index, citation.end_index

        # extract exactly the cited span
        excerpt       = report_text[start:end]

        # extract up to 100 chars immediately before the citation
        pre_start     = max(0, start - 100)
        preceding_txt = report_text[pre_start:start]

        print("MCP CITATION SAMPLE:")
        print(f"  Title:       {citation.title}")
        print(f"  URL:         {citation.url}")
        print(f"  Location:    chars {start}–{end}")
        print(f"  Preceding:   {preceding_txt!r}")
        print(f"  Excerpt:     {excerpt!r}")
        break

print("--------")


# EXAMPLE MCP CITATION

# REPORT EXCERPT:
# # Introduction
# Semaglutide – a glucagon-like peptide-1 (GLP-1) analogue – has rapidly become a blo
# --------
# MCP CITATION SAMPLE:
#   Title:       Document file-WqbCdYNqNzGuFfCAeWyZfp
#   URL:         https://platform.openai.com/storage/files/file-WqbCdYNqNzGuFfCAeWyZfp
#   Location:    chars 237–331
#   Preceding:   'and obesity due to its potent clinical efficacy (often inducing ~10–15% b
#   Excerpt:     '([platform.openai.com](https://platform.openai.com/storage/files/file-Wqb


# print the MCP tool calls
calls = [
    (item.name, item.server_label, item.arguments)
    for item in response.output
    if item.type == "mcp_call" and item.arguments
]
for name, server, args in calls:
    print(f"{name}@{server} → {args}")
print("--------")
```

Clarifying Questions in ChatGPT vs. the Deep Research API

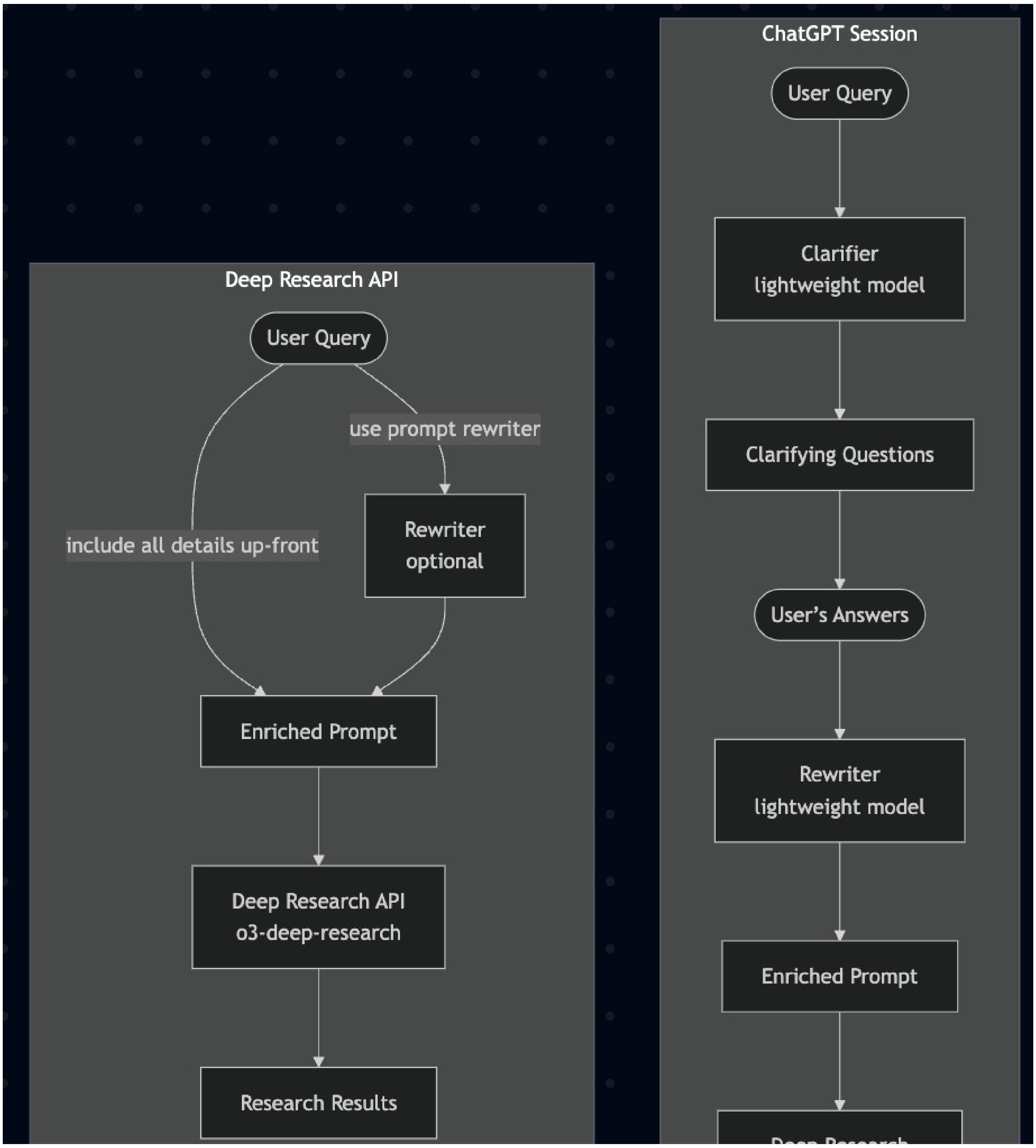
If you've used Deep Research in ChatGPT, you may have noticed that it often asks follow-up questions after you submit a query. This is intentional: ChatGPT uses an intermediate model (like gpt-4.1) to help clarify your intent and gather more context (such as your preferences, goals, or constraints) before the research process begins. This extra step helps the system tailor its web searches and return more relevant and targeted results.

In contrast, the Deep Research API skips this clarification step. As a developer, you can configure this processing step to rewrite the user prompt or ask a set of clarifying questions, since the model expects fully-formed prompts up front and will not ask for additional context or fill in missing information; it simply starts researching based on the input it receives.

To get strong, reliable outputs from the API, you can use two approaches.

- Use a prompt rewriter using another lightweight model (e.g., gpt-4.1) to expand or specify user queries before passing them to the research model.
- Include all relevant details: desired scope, comparisons, metrics, regions, preferred sources, and expected output format.

This setup gives developers full control over how research tasks are framed, but also places greater responsibility on the quality of the input prompt. Here's an example of a generic rewriting_prompt to better direct the subsequent deep research query.



Deep Research
o3-deep-research
Research Results

Here's an example of a rewriting prompt:

```
suggested_rewriting_prompt = """
You will be given a research task by a user. Your job is to produce a set of instructions f

GUIDELINES:
1. **Maximize Specificity and Detail**
- Include all known user preferences and explicitly list key attributes or dimensions to co
- It is of utmost importance that all details from the user are included in the instruction

2. **Fill in Unstated But Necessary Dimensions as Open-Ended**
- If certain attributes are essential for a meaningful output but the user has not provided

3. **Avoid Unwarranted Assumptions**
- If the user has not provided a particular detail, do not invent one.
- Instead, state the lack of specification and guide the researcher to treat it as flexible

4. **Use the First Person**
- Phrase the request from the perspective of the user.

5. **Tables**
- If you determine that including a table will help illustrate, organize, or enhance the in
Examples:
- Product Comparison (Consumer): When comparing different smartphone models, request a tabl
- Project Tracking (Work): When outlining project deliverables, create a table showing task
- Budget Planning (Consumer): When creating a personal or household budget, request a table
Competitor Analysis (Work): When evaluating competitor products, request a table with key m

6. **Headers and Formatting**
- You should include the expected output format in the prompt.
- If the user is asking for content that would be best returned in a structured format (e.g

7. **Language**
- If the user input is in a language other than English, tell the researcher to respond in

8. **Sources**
- If specific sources should be prioritized, specify them in the prompt.
- For product and travel research, prefer linking directly to official or primary websites
- For academic or scientific queries, prefer linking directly to the original paper or offi
- If the query is in a specific language, prioritize sources published in that language.
"""
```

```
response = client.responses.create(
    instructions=suggested_rewriting_prompt,
    model="gpt-4.1-2025-04-14",
    input="help me plan a trip to france",
)
```

```
new_query = response.output[0].content[0].text
print(new_query)
```

In this instance, a user submitted a generic or open-ended query without specifying key details like travel dates, destination preferences, budget, interests, or travel companions; the rewriting prompt rewrote the query so Deep Research will attempt to generate a broad and inclusive response that anticipates common use cases.

While this behavior can be helpful in surfacing a wide range of options, it often leads to verbosity, higher latency, and increased token usage, as the model must account for many possible scenarios. This is especially true for queries that trigger complex planning or synthesis tasks (e.g. multi-destination travel itineraries, comparative research, product selection).

Instead of proceeding immediately with a broad research plan, let's trying using a lighter weight model to gently ask clarification questions from the user before generating a full answer and then using the rewriting prompt for clearer output for the model.

```
suggested_clariying_prompt = """"
You will be given a research task by a user. Your job is NOT to complete the task yet, but

GUIDELINES:
1. **Maximize Relevance**
- Ask questions that are *directly necessary* to scope the research output.
- Consider what information would change the structure, depth, or direction of the answer.

2. **Surface Missing but Critical Dimensions**
- Identify essential attributes that were not specified in the user’s request (e.g., prefer
- Ask about each one *explicitly*, even if it feels obvious or typical.

3. **Do Not Invent Preferences**
```

```
- If the user did not mention a preference, *do not assume it*. Ask about it clearly and ne

4. **Use the First Person**
- Phrase your questions from the perspective of the assistant or researcher talking to the

5. **Use a Bulleted List if Multiple Questions**
- If there are multiple open questions, list them clearly in bullet format for readability.

6. **Avoid Overasking**
- Prioritize the 3–6 questions that would most reduce ambiguity or scope creep. You don’t n

7. **Include Examples Where Helpful**
- If asking about preferences (e.g., travel style, report format), briefly list examples to

8. **Format for Conversational Use**
- The output should sound helpful and conversational—not like a form. Aim for a natural ton
"""



response = client.responses.create(
    instructions=suggested_clariying_prompt,
    model="gpt-4.1-2025-04-14",
    input="help me plan a trip to france",
)

new_query = response.output[0].content[0].text
print(new_query)
```

```
user_follow_up = """I'd like to travel in August. I'd like to visit Paria and Nice. I'd lik
I'm going with my friend. we're both in our mid-twenties. i like history, really good frenc
"""
instructions_for_DR = client.responses.create(
    instructions=suggested_rewriting_prompt,
    model="gpt-4.1-2025-04-14",
    input=user_follow_up,
)
instructions_for_deep_research = instructions_for_DR.output[0].content[0].text
print(instructions_for_deep_research)
```

```
deep_research_call = client.responses.create(
  model="o4-mini-deep-research-2025-06-26",
  input=[
    {
      "role": "developer",
      "content": [
        {
          "type": "input_text",
          "text": instructions_for_deep_research,
        }
      ]
    },
  ],
  reasoning={
    "summary": "auto"
  },
  tools=[
    {
      "type": "web_search_preview"
    },
  ]
)
```

```
# Access the final report from the response object
print(deep_research_call.output[-1].content[0].text)
```

And there you have it! A deep research report crafted for your upcoming trip to France!

In this notebook, we explored how to use the Deep Research API to automate complex, real-world research tasks, from analyzing the economic impact of semaglutide to planning a trip to France that works for you. Deep Research shines when you need structured, citation-backed answers grounded in real-world evidence. Some standout use cases include:

- Product comparisons and market analyses
- Competitive intelligence and strategy reports
- Technical literature reviews and policy synthesis

Whether you're looking to build research agents, generate structured reports, or integrate high-quality synthesis into your workflows, we hope the examples here help you get started.

What's next? Deep Research Agents