AI Programming

데이터 마이닝 / 생성형 AI

08. Openai API

first coding

first coding

머신러닝 개요 및 지도학습 소개

Openai API 소개

Openai API

- OpenAI와 Openai API 개요
  - OpenAI는 인공지능(AI) 연구 및 서비스 기업으로, 대화형 인공지능인 GPT(Generative Pre-trained Transformer) 시리즈를 개발
  - API를 통해 여러 모델을 활용할 수 있도록 지원
  - GPT-4는 기존 모델(GPT-3, GPT-3.5 등) 대비 뛰어난 추론력, 넓은 컨텍스트 처리 능력, 다양한 과업에 적합한 유연성을 갖춘 최신 언어 모델
  - 2023년 6월 업데이트에서 Function Calling(함수 호출) 기능이 도입되어, 외부 도구 연동 및 복잡한 업무 자동화 시 높은 성능을 발휘

Openai API

- **GPT-4 API 작동 원리**
  - GPT-4 API는 RESTful 방식으로 동작하며, 사용자는 API를 통해 프롬프트(입력)를 보내고, 모델이 답변을 생성
  - 대표적으로 ChatCompletion 엔드포인트를 사용하여, 대화형 AI 응답을 받을 수 있음

- **ChatCompletions API의 구조와 역할(role) 개념**
  - ChatCompletion API에서는 대화가 메시지의 리스트로 구성되며, 각 메시지에는 role(역할) 정보가 명시
    - system: 모델의 행동 방향, 기본 지침, 맥락 정보를 설정(예: “너는 친절한 도우미야”)
    - user: 사용자의 질문, 요청, 입력 내용(예: “오늘 날씨 알려줘”)
    - assistant: 모델(GPT-4)이 생성한 응답(예: “오늘 서울의 날씨는 맑음입니다.”)
  - 각 역할의 구분은 대화의 흐름을 자연스럽게 이어가며, 맥락(Context) 유지에 필수적인 요소

Openai API

- **시스템 메시지의 활용**
  - 시스템 메시지를 통해 대화의 톤, 응답 스타일, 지침을 사전에 설정할 수 있습니다.

  예시) “시스템: 너는 유치원 선생님이야! 쉽고 재미있게 설명해야 해”와 같이 설정하면 모델이 그에 맞는 답변을 제공

- **openai 모듈 vs HTTP 직접 호출**
  - openai 패키지는 내부적으로 HTTP 요청을 자동화하여 개발자가 쉽게 사용 가능
  - 별도의 HTTP 클라이언트(requests 등)로 직접 호출도 가능하며, REST API 명세를 그대로 사용할 때 활용
  - 대부분의 경우 openai 모듈이 간편하고 안전함

Python을 활용한 Openai API 호출

- 채팅 응답 구조
  - Responses
    - 모델 응답 생성을 위한 OpenAI의 가장 진보된 인터페이스
    - 텍스트 및 이미지 입력과 텍스트 출력을 지원
    - 이전 응답의 출력을 입력으로 사용하여 모델과 상태 저장 상호 작용
    - 파일 검색, 웹 검색, 컴퓨터 사용 등을 위한 기본 제공 도구로 모델의 기능을 확장
    - 함수 호출을 사용하여 모델이 외부 시스템 및 데이터에 액세스
  - Chat Completions
    - Chat Completions API 엔드포인트는 대화를 구성하는 메시지 목록에서 모델 응답을 생성

Python을 활용한 Openai API 호출

- 로컬 개발 환경 설정
  - api key 설정 : Windows

```
setx OPENAI_API_KEY "your_api_key_here"
```

  - api key 설정 : macOS 또는 Linux

```
export OPENAI_API_KEY="your_api_key_here"
```

  - 공식 SDK 설치

```
pip install openai
```

Python을 활용한 Openai API 호출

- 채팅 응답 구조
  - Responses
    - 모델 응답 생성을 위한 OpenAI의 가장 진보된 인터페이스
    - 텍스트 및 이미지 입력과 텍스트 출력을 지원
    - 이전 응답의 출력을 입력으로 사용하여 모델과 상태 저장 상호 작용
    - 파일 검색, 웹 검색, 컴퓨터 사용 등을 위한 기본 제공 도구로 모델의 기능을 확장
    - 함수 호출을 사용하여 모델이 외부 시스템 및 데이터에 액세스
  - Chat Completions
    - Chat Completions API 엔드포인트는 대화를 구성하는 메시지 목록에서 모델 응답을 생성

Python을 활용한 Openai API 호출

- 기본 API 요청 테스트 : Response API
  - GPT-4와 대화 예시 (https://platform.openai.com/docs/guides/text?api-mode=responses)

```
from openai import OpenAI

client = OpenAI(api_key=openai_api_key)

response = client.responses.create(
    model="gpt-4.1",
    input="Write a one-sentence bedtime story about a unicorn."
)

print(response.output_text)
```

```
Under the soft glow of a silver moon, a gentle unicorn named Luna tiptoed through a
field of twinkling stars, carrying sweet dreams to every sleepy child.
```

Python을 활용한 Openai API 호출

- openai 모듈로 GPT-4 호출 : Response API
  - GPT-4와 대화 예시 (https://platform.openai.com/docs/guides/text?api-mode=responses)

```
response = client.responses.create(
    model="gpt-4.1",
    instructions="Talk like a pirate.",
    input="Are semicolons optional in JavaScript?",
)

print(response.output_text)
```

```
Arrr matey, hoist the Jolly Roger and listen well! In JavaScript, semicolons be like rum on a pirate
ship—ye *can* get by without 'em, but sometimes leavin' 'em out'll scuttle yer code and make it walk the
plank!

JavaScript has a tricksy bit called **Automatic Semicolon Insertion (ASI)**. Most o' the time, she'll
patch in them missing semicolons for ye. But beware! Sometimes the scallywag o' ASI gets it wrong,
'specially after certain lines, and yer code'll behave like a drunken sailor.

So, *technically*—aye, they be "optional" oftentimes. But if ye want smooth sailin' over troubled
waters, drop them semicolons in yerself. It'll save ye from many a cursed bug and stack trace squall.

Arrr, keep yer curly braces close and yer semicolons closer! 🏴‍☠️
```

Python을 활용한 Openai API 호출

- openai 모듈로 GPT-4 호출 : Response API
  - GPT-4와 대화 예시 (https://platform.openai.com/docs/guides/text?api-mode=responses)

```
response = client.responses.create(
    model="gpt-4.1",
    input=[
        {
            "role": "developer",
            "content": "Talk like a pirate."
        },
        {
            "role": "user",
            "content": "Are semicolons optional in JavaScript?"
        }
    ]
)
print(response.output_text)
```

Python을 활용한 Openai API 호출

- inpuit 설정
  - 모델이 역할에 따라 다른 메시지에 어떻게 다른 수준의 우선순위를 부여하는지 설명

developer	user	assistant
개발자 메시지는 애플리케이션 개발자가 제공하는 지침으로, 사용자 메시지보다 우선순위가 높습니다.	사용자 메시지는 최종 사용자가 제공하는 지침으로, 개발자 메시지 뒤로 우선순위가 정해집니다.	모델에 의해 생성된 메시지는 보조 역할을 합니다.

  - developer 및 user 메시지는 프로그래밍 언어의 함수와 그 인수처럼 생각할 수 있습니다.
  - developer 메시지는 함수 정의처럼 시스템의 규칙과 비즈니스 로직을 제공
  - user 메시지는 함수에 대한 인수처럼 개발자 메시지 지침이 적용되는 입력 및 구성을 제공

Python을 활용한 Openai API 호출

- **재사용 가능한 프롬프트**
  - OpenAI 대시보드에서는 코드에서 프롬프트 내용을 지정하는 대신, API 요청에 사용할 수 있는 재사용 가능한 프롬프트를 개발 기능 제공
  - 데시보드에서 프롬프트 생성 : https://platform.openai.com/playground/prompts?models=gpt-4.1
  - 프롬프트 매개변수 객체에는 설정
    - Id : 대시보드에서 찾을 수 있는 프롬프트의 고유 식별자
    - version : 프롬프트의 특정 버전(대시보드에 지정된 "현재" 버전으로 기본 설정됨)
    - variables : 프롬프트에서 변수를 대체할 값의 map 입니다.
      대체 값은 문자열이거나 input_image 또는 input_file 와 같은 다른 응답 입력
    - 상세 API : https://platform.openai.com/docs/api-reference/responses/create

Python을 활용한 Openai API 호출

- 재사용 가능한 프롬프트
  - 예시코드

```
response = client.responses.create(
    model="gpt-4.1",
    prompt={
        "id": "pmpt_abc123",
        "version": "2",
        "variables": {
            "customer_name": "Jane Doe",
            "product": "40oz juice box"
        }
    }
)

print(response.output_text)
```

Python을 활용한 Openai API 호출

- openai 모듈로 GPT-4 호출 : Chat Completions API
  - GPT-4와 대화 예시 (https://platform.openai.com/docs/guides/text?api-mode=chat)

```
completion = client.chat.completions.create(
    model="gpt-4.1",
    messages=[
        {
            "role": "user",
            "content": "Write a one-sentence bedtime story about a unicorn."
        }
    ]
)

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

Python을 활용한 Openai API 호출

- openai 모듈로 GPT-4 호출 : Chat Completions API
  - Choices 응답 속성 구조

```
[
    {
        "index": 0,
        "message": {
            "role": "assistant",
            "content": "Under the soft glow of the moon, Luna the unicorn
danced through fields of twinkling stardust, leaving trails of dreams for
every child asleep.",
            "refusal": null
        },
        "logprobs": null,
        "finish_reason": "stop"
    }
]
```

Python을 활용한 Openai API 호출

- 콘솔에서 사용자 입력(질문, 프롬프트)를 받고 응답 처리하는 프로그램을 만들어 봅시다.

Function Calling 개념

- **Function Calling의 등장 배경**
  - 기존 GPT 기반 챗봇은 사용자의 질문에 대해 텍스트 답변만 생성
  - 구조화된 데이터 반환, 외부 시스템 연동 등에서는 추가 파싱, 별도 규칙 적용 등 복잡한 처리가 필요
  - 실제 외부 도구(API, DB 등)와 신뢰성 있게 연동하는 데 한계

- **Function Calling이란?**
  - Function Calling은 GPT-4의 새로운 기능으로, 모델이 함수 호출 형태의 응답을 생성해 외부 시스템 또는 API와 직접 연동할 수 있게 함
  - 개발자가 정의한 함수의 스펙(함수 이름, 설명, 파라미터 등)에 따라, 모델이 JSON 형식의 인자를 생성하여 함수 호출 요청을 반환

Function Calling 개념

- **Function Calling 동작 흐름**

① 사용자 질문과 함수 목록 전달

- 애플리케이션이 사용자 메시지와 함께 호출 가능한 함수 목록(functions 파라미터)을 모델에 전달

② 모델의 응답 판단

- 모델은 사용자 요청을 해석하여 답변 또는 함수 호출 중 적합한 방법을 선택
- 함수 설명을 기반으로 호출이 필요한 상황을 스스로 감지

③ 함수 호출 요청 반환

- 모델이 함수가 필요하다고 판단하면, 호출할 함수명과 인자를 JSON으로 담은 function_call 응답 반환
- 예시: {"role": "assistant", "function_call": {"name": "getWeather", "arguments": "{...}"}}

Function Calling 개념

- Function Calling 동작 흐름

④ 함수 실행 및 결과 반환

- 애플리케이션(백엔드)이 실제 함수를 실행하고, 결과를 function 역할 메시지로 다시 모델에 전달
- 예시: {"role": "function", "name": "getWeather", "content": "맑음, 23도"}

⑤ 최종 응답 생성

- 모델은 대화 내력과 함수 실행 결과를 바탕으로 최종 사용자 답변 생성
- 필요에 따라 추가 함수 호출도 가능

Function Calling 개념

- **Function Calling의 주요 활용 사례 및 효과**
  - 자연어 명령을 자동으로 외부 API 호출, 데이터베이스 질의 등으로 변환 가능
  - 텍스트에서 구조화된 데이터 추출 및 다양한 외부 시스템과의 통합이 신뢰성 있게 가능플러그인 기반 챗봇, 자동화 에이전트 등 실질적인 실무 적용 폭이 대폭 확대됨
- **기존 챗봇과의 차이점**
  - 기존 챗봇: 텍스트 생성 후, 별도 파싱 및 룰 엔진 필요(복잡함, 신뢰성 부족)
  - Function Calling: 함수 호출 요청과 결과 처리를 대화 흐름 내에서 명확하게 관리(확장성, 신뢰성 우수)
  - 별도의 파인튜닝(Fine-tuning)이나 복잡한 프롬프트 엔지니어링 없이 기능 확장 가능
- **구현의 실제 책임 구분**
  - GPT-4 모델은 함수 호출 "요청"만 생성
  - 실제 함수 실행 및 결과 반영(로직)은 반드시 개발자/애플리케이션 측이 수행
  - 모델이 직접 코드를 실행하거나 시스템에 영향을 주지 않음에 유의

Built-in tools

- 내장 도구의 사용
  - 원격 MCP 서버 또는 웹 검색과 같은 도구를 사용하여 모델의 기능을 확장할 수 있음
  - 페이지 복사하기 모델 응답을 생성할 때 기본 제공 도구를 사용하여 모델 기능을 확장할 수 있음
  - 이러한 도구는 모델이 웹이나 파일에서 추가 컨텍스트 및 정보에 액세스하는 데 도움

Built-in tools

- 사용 가능한 도구

  - **함수 호출** : 사용자 지정 코드를 호출하여 모델에 추가 데이터 및 기능에 대한 액세스 권한을 부여
  - **웹 검색** : 모델 응답 생성에 인터넷의 데이터를 포함
  - **원격 MCP 서버** : 모델 컨텍스트 프로토콜(MCP) 서버를 통해 모델에 새로운 기능에 대한 액세스 권한을 부여
  - **파일 검색** : 응답을 생성할 때 업로드된 파일의 내용을 검색하여 컨텍스트를 확인
  - **이미지 생성** : GPT 이미지를 사용하여 이미지를 생성하거나 편집
  - **코드 인터프리터** : 모델이 보안 컨테이너에서 코드를 실행할 수 있도록 허용
  - **컴퓨터 사용** : 모델이 컴퓨터 인터페이스를 제어할 수 있는 에이전트 워크플로를 생성

Built-in tools

- 사용 가능한 도구: 웹검색 예제
  - Include web search results for the model response

```
response = client.responses.create(
    model="gpt-4.1",
    tools=[{"type": "web_search_preview"}],
    input="What was a positive news story from today?"
)

print(response.output_text)
```

Built-in tools

- **API에서의 사용법**
  - **모델 응답을 생성하도록 요청할 때 도구 매개변수에 구성을 지정하여 도구 액세스를 사용 설정**
  - 각 도구마다 고유한 구성 요구 사항이 있으며, 자세한 지침은 사용 가능한 도구 섹션을 참조
  - 제공된 프롬프트에 따라 모델은 구성된 도구를 사용할지 여부를 자동으로 결정
    예를 들어,
    프롬프트에서 모델의 학습 마감일 이후의 정보를 요청하고 웹 검색이 활성화된 경우, 모델은 일반적으로 웹 검색 도구를 호출하여 관련 최신 정보를 검색
  - API 요청에서 tool_choice 매개변수를 설정하여 이 동작을 명시적으로 제어하거나 안내할 수 있음
- **함수 호출**
  - 기본 제공 도구 외에도 도구 배열을 사용하여 사용자 지정 함수를 정의
  - 사용자 지정 함수를 사용하면 모델에서 애플리케이션의 코드를 호출하여 모델 내에서 직접 사용할 수 없는 특정 데이터나 기능에 액세스할 수 있음 → **머신러닝으로 학습한 모델을 이용하는 함수를 호출해서 사용!**

Built-in tools

- **함수호출 예제**
  - Function Calling(함수 호출) 기능에서 사용할 툴(함수)을 정의

```
tools = [{
    "type": "function",
    "name": "get_weather",
    "description": "Get current temperature for provided coordinates in celsius.",
    "parameters": {
        "type": "object",
        "properties": {
            "latitude": {"type": "number"},
            "longitude": {"type": "number"}
        },
        "required": ["latitude", "longitude"],
        "additionalProperties": False
    },
    "strict": True
}]
```

Built-in tools

- **함수호출 예제**
  - Function Calling(함수 호출) 기능에서 사용할 툴(함수)을 정의
    - **tools = [{ ... }]**

      여러 개의 함수 정의를 리스트로 묶어서 전달
    - **"type": "function"**

      함수 호출용 툴임을 명시, 만약 이미지를 생성하는 툴이면 "type": "image"처럼 다르게 표기
    - **"name": "get_weather"**

      함수의 이름, 나중에 모델이 함수 호출 요청 시 "get_weather"라는 이름으로 호출
    - **"description": "Get current temperature for provided coordinates in celsius."**

      이 함수가 어떤 역할을 하는지 설명하는 텍스트, 모델이 함수 선택/이해 시 참고
    - **"parameters": { ... }**

      함수에 전달해야 할 파라미터(입력값) 정의
    - **"strict": True**

      엄격하게 파라미터 형식이 맞는 경우에만 함수 호출을 허용, 위도/경도 중 하나라도 빠지면 함수 호출이 실행되지 않음

Built-in tools

- **함수호출 예제**
  - Function calling example with get_weather function

```
input_messages = [{"role": "user", "content": "What's the weather like in Paris today?"}]

response = client.responses.create(
    model="gpt-4.1",
    input=input_messages,
    tools=tools,
)

print('1. ', response.output)
```

Built-in tools

- **함수호출 예제**
  - 결과

```
1.  [ResponseFunctionToolCall(
        arguments='{"latitude":48.8566,"longitude":2.3522}’,
        call_id='call_6BUu8Eq2gxDIxJqptOvTGUN0’,
        name='get_weather’,
        type='function_call’,
        id='fc_6856e0a0ae0c819d8b0b129b4e2f413106e90f960fdda7cf’,
        status='completed’)
]
```

Built-in tools

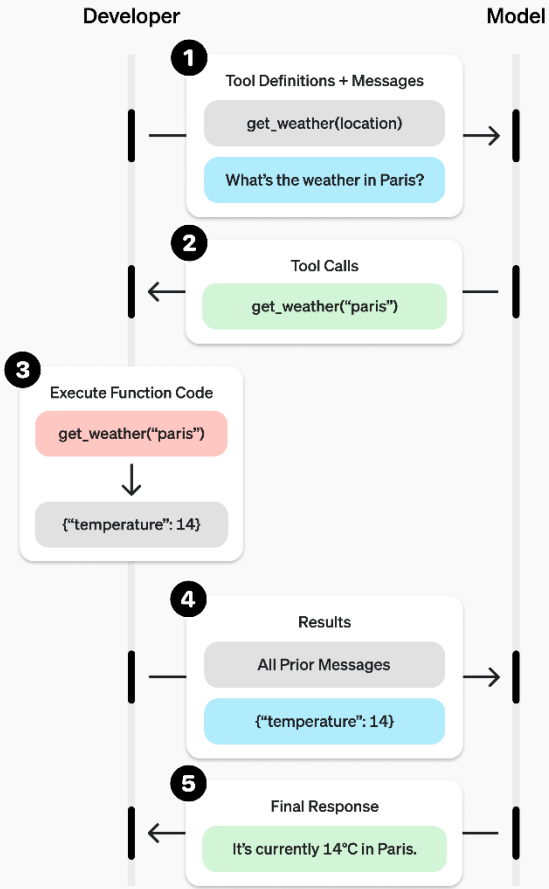
① **툴(함수) 정의 및 메시지 입력**
- 개발자가 사용할 함수(예: get_weather(location))를 정의
- 사용자가 챗봇에 질문을 입력. 예시: “파리의 날씨가 어때?”

② **툴(함수) 호출 요청**
- 모델이 사용자의 질문에서 함수가 필요하다고 판단하면, 해당 함수 호출을 요청
- 예시: get_weather("paris") 함수 호출

③ **함수 코드 실행**
- 개발자가 실제로 함수 코드를 실행
- 함수 실행 결과를 받아옴. 예시: { "temperature": 14 } (파리의 온도 14도)

④ **모델에게 실행 결과 전달**
- 실행 결과(함수 반환값)를 모델에 다시 전달.
- 모델은 이전 대화 내용과 함께 결과 값을 분석.
  예시: { "temperature": 14 } 값과 모든 이전 메시지 전달

⑤ **최종 응답 생성**
- 모델이 모든 정보를 바탕으로 최종 사용자에게 보여줄 답변을 생성
- 예시: “현재 파리의 온도는 14°C입니다.”



Built-in tools

- 함수호출 예제
  - Function calling example with get_weather function

```
import json


tool_call = response.output[0]
args = json.loads(tool_call.arguments)

result = get_weather(args["latitude"], args["longitude"])
print('2. ', result)
```

Built-in tools

- 함수호출 예제
  - 결과

```
2.  34.8
```

Built-in tools

- **함수호출 예제**
  - Function calling example with get_weather function

```
input_messages.append(tool_call)  # append model's function call message

input_messages.append({                                # append result message
    "type": "function_call_output",
    "call_id": tool_call.call_id,
    "output": str(result)
})

response_2 = client.responses.create(
    model="gpt-4.1",
    input=input_messages,
    tools=tools,
)

print(response_2.output_text)
```

Built-in tools

- **함수호출 예제**
  - 결과

```
The weather in Paris today is quite warm, with a temperature of 34.8°C. If you plan to go
outside, make sure to stay hydrated and protect yourself from the sun!
```

Built-in tools

- 함수호출 하는 챗봇 구성
- 임의의 함수를 정의하고 설정해서 응답 메시지 결과에 반영이 되도록 정의
  - 나이 계산 함수

    예시: "1992-05-01생의 만 나이는?

    함수명: calculate_age 입력: birthdate(문자열, YYYY-MM-DD)
  - 환율 변환 함수

    예시: "100달러를 원화로 바꿔줘 (환율 1330원 적용)"

    함수명: convert_currency 입력: amount(숫자), from_currency(문자열), to_currency(문자열), rate(숫자)
  - BMI(체질량지수) 계산 함수

    예시: "키 170cm, 몸무게 65kg의 BMI는?“

    함수명: calculate_bmi 입력: height(숫자, cm), weight(숫자, kg)

Built-in tools

- **OpenAI ChatCompletions API를 호출하는 래퍼 함수 구현**
  - 챗봇/에이전트에서 대화 흐름의 중심 역할
  - 주요 인자: messages(메시지 리스트), functions(함수 목록), function_call 옵션 등
  - 모델의 응답(response)을 받아서 적절히 해석/처리하는 부분이 핵심

```
def call_openai(messages, functions=None, function_call="auto"):

    response = openai.ChatCompletion.create(
        model="gpt-4o",  # 또는 gpt-4-turbo
        messages=messages,
        functions=functions,
        function_call=function_call
    )

    return response
```

Built-in tools

- **OpenAI ChatCompletions API를 호출하는 래퍼 함수**
  - 주요 파라미터
    - messages: 대화의 모든 메시지 이력(역할별 developer, user, assistant, function 등)
    - functions: 함수 정의(JSON Schema) 목록, Function Calling 지원 시 필수
    - function_call: 함수 호출 동작 방식 지정
      - "auto"(기본값): 모델이 필요하다고 판단될 때만 함수 호출
      - "none": 함수 호출을 비활성화(오직 텍스트 응답만)
      - {"name": "<함수명>"}: 특정 함수만 강제로 호출하도록 지정

Built-in tools

- **OpenAI ChatCompletions API를 호출하는 래퍼 함수**
  - 주요 파라미터
    - messages: 대화의 모든 메시지 이력(역할별 developer, user, assistant, function 등)
    - functions: 함수 정의(JSON Schema) 목록, Function Calling 지원 시 필수
    - function_call: 함수 호출 동작 방식 지정
      - "auto"(기본값): 모델이 필요하다고 판단될 때만 함수 호출
      - "none": 함수 호출을 비활성화(오직 텍스트 응답만)
      - {"name": "<함수명>"}: 특정 함수만 강제로 호출하도록 지정

Built-in tools

- Web search
  - 모델이 응답을 생성하기 전에 웹에서 최신 정보를 검색할 수 있도록 허용
  - Responses API를 사용하면 API 요청 시 도구 배열에서 웹 검색을 구성하여 콘텐츠를 생성할 수 있음
  - 다른 도구와 마찬가지로 모델은 입력 프롬프트의 내용에 따라 웹 검색을 수행할지 여부를 선택할 수 있음

```
response = client.responses.create(
    model="gpt-4.1",
    tools=[{"type": "web_search_preview"}],
    input="What was a positive news story from today?"
)

print(response.output_text)
```

  - tool_choice 매개변수를 사용하여 web_search_preview 도구의 사용을 강제할 수 있음
  - 이를 {type: "web_search_preview"}로 설정하면 지연 시간을 줄이고 더 일관된 결과를 얻을 수 있음

Built-in tools

- Web search
  - 출력 및 인용
    - 웹 검색 도구를 사용하는 모델 응답은 두 부분으로 구성
      - 웹 검색 호출의 ID를 포함하는 web_search_call 출력 항목.
      - 메시지 출력 항목에는 다음과 같은 내용이 포함됩니다:
        - » 메시지 콘텐츠의 텍스트 결과(message.content[0].text)
        - » 인용된 URL에 대한 주석(message.content[0].annotations)
    - 기본적으로 모델의 응답에는 웹 검색 결과에서 발견된 URL에 대한 인라인 인용이 포함
    - url_citation 주석 객체에는 인용된 소스의 URL, 제목 및 위치가 포함

Built-in tools

- Web search
  - 출력 및 인용

```
[
  { "type": "web_search_call",
    "id": "ws_67c9fa0502748190b7dd390736892e100be649c1a5ff9609",
    "status": "completed" },
  {
    "id": "msg_67c9fa077e288190af08fdffda2e34f20be649c1a5ff9609",
    "type": "message", "status": "completed", "role": "assistant",
    "content": [
      {
        "type": "output_text",
        "text": "On March 6, 2025, several news...",
        "annotations": [
          {
            "type": "url_citation",
            "start_index": 2606,
            "end_index": 2758,
            "url": "https://...",
            "title": "Title..."
          }…
```

Built-in tools

- Web search
  - User location
    - 지리적 위치에 따라 검색 결과를 필터링하려면 국가, 도시, 지역 및/또는 시간대를 사용하여 대략적인 사용자 위치를 지정
    - 도시 및 지역 필드는 자유 텍스트 문자열로, 각각 Minneapolis와 Minnesota와 같음
    - 국가 필드는 두 글자 ISO 국가 코드로, 예를 들어 US와 같음
    - 시간대 필드는 IANA 시간대 코드로, 예를 들어 America/Chicago와 같음

```
tools=[{
    "type": "web_search_preview",
    "user_location": {
        "type": "approximate",
        "country": "GB",
        "city": "London",
        "region": "London",
    }
}],
```

Built-in tools

- Web search
  - **검색 컨텍스트 크기**
    - 이 도구를 사용할 때 search_context_size 매개변수는 웹에서 검색하여 도구가 응답을 구성하는 데 사용하는 컨텍스트의 양을 제어
    - 검색 도구가 사용하는 토큰은 응답 생성 요청에서 모델 매개변수로 지정된 주요 모델의 컨텍스트 창에 영향을 미치지 않음
    - 이러한 토큰은 한 번의 대화 단계에서 다음 단계로 전달되지 않음
    - 단순히 도구의 응답을 구성하는 데 사용된 후 버려짐

Built-in tools

- Web search
  - **컨텍스트 크기를 선택**
    - **비용**

      검색 도구 가격은 이 매개변수의 값에 따라 달라짐. 더 큰 컨텍스트 크기는 더 높은 비용이 발생
    - **품질**

      더 큰 검색 컨텍스트 크기는 일반적으로 더 풍부한 컨텍스트를 제공하여 더 정확하고 포괄적인 답변을 생성
    - **지연 시간**

      더 큰 컨텍스트 크기는 더 많은 토큰을 처리해야 하므로 도구 응답 시간이 느려질 수 있음

Built-in tools

- Web search
  - 컨텍스트 크기 옵션
    - high

      가장 포괄적인 컨텍스트, 가장 높은 비용, 가장 느린 응답.
    - medium (기본)

      컨텍스트, 비용, 지연 시간의 균형 잡힌 조합.
    - low

      가장 적은 컨텍스트, 가장 낮은 비용, 가장 빠른 응답이지만 답변 품질이 낮을 수 있음

  - 검색 도구가 사용하는 토큰은 주요 모델의 토큰 사용량에 영향을 주지 않으며, 턴 간에 전달되지 않음

Built-in tools

- Web search
  - Customizing search context size

```
response = client.responses.create(
    model="gpt-4.1",
    tools=[{
        "type": "web_search_preview",
        "search_context_size": "low",
    }],
    input="What movie won best picture in 2025?",
)

print(response.output_text)
```

Built-in tools

- File search
  - 모델이 응답을 생성하기 전에 파일에서 관련 정보를 검색할 수 있도록 허용
  - 파일 검색은 Responses API에서 사용할 수 있는 도구
  - **이 도구는 모델이 이전에 업로드된 파일의 지식 기반에서 의미론적 및 키워드 검색을 통해 정보를 검색**
  - **벡터 저장소를 생성**하고 **파일을 업로드**함으로써, 모델의 내재된 지식을 확장하기 위해 이러한 지식 기반이나 벡터 저장소에 접근할 수 있음
  - 이 도구는 OpenAI에서 호스팅 및 관리되는 도구이므로, 실행을 처리하기 위해 코드를 구현할 필요가 없음
  - 모델이 이 도구를 사용하기로 결정하면 자동으로 도구를 호출하여 파일에서 정보를 검색하고 출력을 반환

Built-in tools

- File search
  - Responses API를 사용하여 파일 검색을 수행하기 전에, 벡터 스토어에 지식 기반을 설정하고 해당 스토어에 파일을 업로드해야 함

```
import requests
from io import BytesIO
from openai import OpenAI

client = OpenAI()

def create_file(client, file_path):
    if file_path.startswith("http://") or file_path.startswith("https://"):
        # Download the file content from the URL
        response = requests.get(file_path)
        file_content = BytesIO(response.content)
        file_name = file_path.split("/")[-1]
        file_tuple = (file_name, file_content)
        result = client.files.create(
            file=file_tuple,
            purpose="assistants"
        )
```

Built-in tools

- File search
  - Responses API를 사용하여 파일 검색을 수행하기 전에, 벡터 스토어에 지식 기반을 설정하고 해당 스토어에 파일을 업로드해야 함

```
    else:
        # Handle local file path
        with open(file_path, "rb") as file_content:
            result = client.files.create(
                file=file_content,
                purpose="assistants"
            )
    print(result.id)
    return result.id

# Replace with your own file path or URL
file_id = create_file(client, "https://cdn.openai.com/API/docs/deep_research_blog.pdf")
```

Built-in tools

- File search
  - vector store 생성

```
vector_store = client.vector_stores.create(
    name="knowledge_base"
)
print(vector_store.id)
```

Built-in tools

- File search
  - 파일을 vector store에 추가

```
client.vector_stores.files.create(
    vector_store_id=vector_store.id,
    file_id=file_id
)

print(result)
```

Built-in tools

- File search
  - 상태 확인 : 파일이 사용 가능해질 때까지(즉, 상태가 완료될 때까지) 실행

```
result = client.vector_stores.files.list(
    vector_store_id=vector_store.id
)

print(result)
```

Built-in tools

- File search
  - 지식 기반이 설정되면, 모델에 사용할 수 있는 도구 목록에 file_search 도구를 포함시킬 수 있으며, 검색할 벡터 저장소 목록도 함께 포함시킬 수 있음

```
response = client.responses.create(
    model="gpt-4o-mini",
    input="What is deep research by OpenAI?",
    tools=[{
        "type": "file_search",
        "vector_store_ids": ["<vector_store_id>"]
    }]
)

print(response)
```

Built-in tools

- File search
  - 검색 결과 맞춤 설정 : **검색 결과 수 제한**
    - Responses API와 파일 검색 도구를 함께 사용하면 벡터 저장소에서 검색하려는 결과 수를 맞춤 설정
    - 토큰 사용량과 지연 시간을 줄이는 데 도움이 될 수 있지만, 답변 품질이 저하될 수 있음

```
response = client.responses.create(
    model="gpt-4o-mini",
    input="What is deep research by OpenAI?",
    tools=[{
        "type": "file_search",
        "vector_store_ids": ["<vector_store_id>"],
        "max_num_results": 2
    }]
)
print(response)
```

Built-in tools

- File search
  - 검색 결과 맞춤 설정 : **응답에 검색 결과를 포함**
    - 출력 텍스트에서 주석(파일 참조)을 확인할 수 있지만, 파일 검색 호출은 기본적으로 검색 결과를 반환하지 않음
    - 응답에 검색 결과를 포함하려면 응답을 생성할 때 include 매개변수를 사용할 수 있습니다.

```
response = client.responses.create(
    model="gpt-4o-mini",
    input="What is deep research by OpenAI?",
    tools=[{
        "type": "file_search",
        "vector_store_ids": ["<vector_store_id>"]
    }],
    include=["file_search_call.results"]
)
print(response)
```

Built-in tools

- File search
  - 검색 결과 맞춤 설정 : **응답에 검색 결과를 포함**
    - 출력 텍스트에서 주석(파일 참조)을 확인할 수 있지만, 파일 검색 호출은 기본적으로 검색 결과를 반환하지 않음
    - 응답에 검색 결과를 포함하려면 응답을 생성할 때 include 매개변수를 사용할 수 있습니다.

```
response = client.responses.create(
    model="gpt-4o-mini",
    input="What is deep research by OpenAI?",
    tools=[{
        "type": "file_search",
        "vector_store_ids": ["<vector_store_id>"]
    }],
    include=["file_search_call.results"]
)
print(response)
```

Built-in tools

- handle_function_call() 메서드 구성
  - 개요
    - 모델 응답이 함수 호출(function_call)인 경우를 처리하는 핵심 로직
    - function_call 정보를 읽어 실제 Python 함수를 실행
    - 그 결과를 새로운 메시지로 messages 리스트에 추가

Built-in tools

- handle_function_call() 메서드 구성

① function_call 필드 확인

- 모델 응답의 message["function_call"] 존재 여부 확인
- 호출할 함수명(name), 인자(arguments) 추출 (arguments는 JSON 문자열)

② 인자 파싱

- arguments는 JSON 형식의 문자열 → Python 딕셔너리로 변환 필요
- json.loads()로 변환 (예외 처리: try-except로 JSONDecodeError 등 방지)

③ 함수 실행 분기

- function_call["name"]에 따라 실제 Python 함수 호출
- 예: name == "add_numbers"이면, add_numbers(**args) 실행
- 다른 함수(예: 날짜 변환)도 동일하게 분기 처리

Built-in tools

- handle_function_call() 메서드 구성

  ④ 함수 실행 결과 메시지 생성

  - 함수 실행 결과(result)를 문자열로 변환
  - 새로운 메시지: { "role": "function", "name": 함수이름, "content": result }

  ⑤ messages 리스트에 추가

  - 생성한 function 메시지를 기존 messages에 append하여 대화 흐름 유지

Built-in tools

- handle_function_call() 메서드 구성

```
import json

def handle_function_call(function_call, messages):
    try:
        func_name = function_call["name"]
        args = json.loads(function_call["arguments"])
    except (KeyError, json.JSONDecodeError):
        result = "함수 호출 인자 오류"
        func_name = function_call.get("name", "unknown")
    else:
        if func_name == " 사용자저의함수1 ":
            result = str(add_numbers(**args))
        elif func_name == " 사용자저의함수2 ":
            result = convert_date(**args)
        else:
            result = "알 수 없는 함수 호출"
    function_message = {
        "role": "function",
        "name": func_name,
        "content": result
    }
    messages.append(function_message)
    return messages
```

Built-in tools

- **통합 함수 chat()**
  - 통합 함수(chat)의 역할
    - chat() 메서드는 사용자 입력부터 최종 답변까지 전체 대화 사이클을 관리하는 통합 함수
    - 메시지 추가, 모델 호출, 함수 실행, 응답 반환까지 일련의 흐름을 담당

Built-in tools

- **통합 함수 chat()**
  - 주요 단계별 동작 순서
    - ① 사용자 입력 추가
      - 사용자 질문을 {"role": "user", "content": 질문} 형태로 messages 리스트에 append
      - 첫 호출이라면 system 메시지도 함께 포함
    - ② 모델 1차 응답 요청
      - call_openai(messages, functions, function_call="auto") 호출
      - 모델 응답(response)에서 첫 번째 메시지 추출
    - ③ function_call 분기 처리
      - 모델 응답에 function_call이 포함된 경우:
        - » handle_function_call()로 해당 함수 실행 및 결과 메시지(messages에 추가)
        - » 함수 결과가 추가된 messages로 call_openai()를 다시 호출해 최종 답변 생성
      - function_call이 없는 일반 답변이라면 바로 사용자에게 반환
    - ④ 최종 답변 반환 : 모델의 최종 assistant 답변(자연어 content)을 반환 또는 출력

Built-in tools

- 통합 함수 chat()

```
def chat(user_input, messages, functions):

    messages.append({"role": "user", "content": user_input}) # 1. 사용자 메시지 추가

    response = call_openai(messages, functions, function_call="auto") # 2. 모델 1차 응답
    reply = response.choices[0].message

    if "function_call" in reply: # 3. 함수 호출 여부 분기
        # 함수 호출 처리
        handle_function_call(reply["function_call"], messages)
        # 함수 실행 결과가 포함된 messages로 다시 모델 호출
        response2 = call_openai(messages, functions)
        final_reply = response2.choices[0].message
        return final_reply.get("content", str(final_reply))
    else:
        # 일반 답변 바로 반환
        return reply.get("content", str(reply))
```

Built-in tools

- 통합 함수 chat()

```
def chat(user_input, messages, functions):

    messages.append({"role": "user", "content": user_input}) # 1. 사용자 메시지 추가

    response = call_openai(messages, functions, function_call="auto") # 2. 모델 1차 응답
    reply = response.choices[0].message

    if "function_call" in reply: # 3. 함수 호출 여부 분기
        # 함수 호출 처리
        handle_function_call(reply["function_call"], messages)
        # 함수 실행 결과가 포함된 messages로 다시 모델 호출
        response2 = call_openai(messages, functions)
        final_reply = response2.choices[0].message
        return final_reply.get("content", str(final_reply))
    else:
        # 일반 답변 바로 반환
        return reply.get("content", str(reply))
```