OpenAI. Documentations - Get Started, Capabilities, Assistant

Get Started

Introduction

OpenAI API 에 관한 개괄적인 설명.

- 관련 개념에 대한 서술도 해당 항목에 잘 되어 있음.

Key Concepts

Text generation models

- (Generative pre-trained transformers or "GPT" models) GPT 는 OpenAI의 텍스트 생성 모델 (Text generation models) 로서, 자연어 및 공식 언어를 이해하도록 훈련되었음.
- 입력에 대한 응답으로 텍스트 출력을 허용.
- 입력은 "프롬프트" 라 함. (관련 문서 : Prompt Engineering 이란 무엇인가)
    - 프롬프트는 GPT 를 프로그래밍 하는 방법 (출력을 조정하는 방법)

Assistants

- 어시스턴트는 OpenAI API의 경우 GPT-4와 같은 대규모 언어 모델로 구동됨.
- Assistants 란, 사용자를 위한 작업을 수행할 수 있는 개체 (Entities) 임.
    - context window 등을 기반으로 embeding 된 지시를 기반으로 수행.
    - 이에 관한 자세한 문서 : Assistants overview - OpenAI API

Embeddings

- Embedding = contents 와 그것의 의미의 관점을 유지하기 위한 data 조각의 vector 표현 (representation).
    - 이는 즉, 임베딩이란, AI 가 일관성을 유지하기 위한 데이터 덩어리이고. 그 데이터 덩어리는 벡터로 표현된다는 뜻이 아닌가.
- 텍스트 문자열을 입력으로 사용하고 임베딩 벡터를 출력으로 생성하는 텍스트 임베딩 모델을 제공 = 텍스트 입력을 하면, AI 가 편향을 가지도록 유도할 수 있는 데이터 조각을 제공한다는 뜻인 듯?

Token

- 모델은 텍스트를 token이라 부르는 chunks (큰 덩어리) 안에서 생성한다.
- token 은 일반적으로 생성되는 문자열의 묶음을 의미한다.
    - ex 1 - "tokenization"은 "token" 및 "ization"으로 분해
    - ex 2 - "the"와 같은 짧고 일반적인 단어는 단일 토큰으로 표시

Quick Start

- OpenAI 는 우리의 앱에서 지능화된 부분을 생성할 수 있는 간단한 interface 를 제공함. 이는 OpenAI 의 최첨단 모델에 기반함.
- 만약 숙련된 개발자라면, API reference / GPT guide 를 바로 살펴봐도 좋음.
- 여기 Quick Start 목차는 아래 내용을 포함.
    1. 개발 환경 설정
    2. 최신 SDK 설치
    3. OpenAI 의 API 개념 중 일부
    4. 첫 번째 API 요청을 보내는 방법

1. 계정 만들기
2. API Key page 에서, "Create new secret key" 클릭.
3. 이후, 각 언어별로 어떻게 진행하는지가 나타나있는데, C# 이나 그냥 API Request 보내는 것에 대해서는 나와 있지 않음.
    - 이 부분은 내가 직접 탐색하는 것이 좋을 듯.

추가 내용

- 모델 및 API에 대한 자세한 내용은 Text generation - OpenAI API 이 부분 부터 참고.
- OpenAI Cookbook에서 심층적 예제 확인 가능.
- 모델이 무엇을 할 수 있는지에 대해선 example prompts 확인.
- 코드를 작성하지 않고 API를 사용하고 싶은 경우, Playground 이용.
- 구축 시작할 경우, usage policies 를 확인.

이외 내용

- Models 의 구성.
- Whisper 및 이외 기타 튜토리얼
- ChangeLogs 가 있는데, 딱히 중요하지 않을 듯.

Capabilities (역량)

> 여기부터 배웠을 때 꽤나 심층적인 내용인 듯. Fine Tuning 에 대한 내용 또한 나와 있기 때문에, 그래도 깊이가 꽤 됨.

Dalle-3 / Vision (GPT 의 이미지 보는 기능) / TTS / STT (Whisper) / Moderation (정책 준수 사항 확인) 과 같은 다양한 사항들이 존재함. 또한, 해당 기능들이 GPT API 호출에 통합되어 있는 점이 기대할만한 점인 듯. 단, 앞에서 언급한 기능의 경우 따로 필요하지 않기 때문에 아래 나열한 항목들에 대해서만 살펴봄.

- Text Generation
- Function Calling
- Embeddings
- Fine-tuning

Text Generation Models

- OpenAI 의 Text Model (GPT), Prompt 에 대한 개괄적인 설명, GPT 를 통해 이룰 수 있는 기능들에 대한 간략한 설명. (위의 Key Concept 에서 이미 설명을 들었던 내용들임! 따라서 간략하게만 짚고 넘어감.)
- API 와 관련해서, API key 와 Value 가 포함되어 있는 Request 를 보내고, Model 로부터 output 된 내용을 response 로 받게 됨. Model 은 chat completion api enpoint 라는 곳을 통해 접근 받게 됨. (*Q. Chat Completion 이라는 개념이 조금씩 계속 등장하고 있는데, 이것이 의미하는 바가 무엇일까? => A. 이거 바로 다음 항목에 나옴.*)

1. Chat Completions API

- 일반적으로 OpenAI 의 GPT 를 사용하기 위해 이용하는 API 로서, 대화 완료 정보를 전송 및 요청 (Request), 이에 따라 GPT 가 생성한 반응을 반환 (Response) 받는다.
  - 이러한, 대화 완료 정보를 전송하고, 이에 따라 GPT 가 반응을 생성한 내용을 반환 받는 형태의 API 를 **Chat Completions API** 라 한다.
- Python 을 이용한 message 전송의 예시는 다음과 같다.

PYTHON

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

response = client.chat.completions.create(
  model="gpt-3.5-turbo",
  messages=[
    {"role": "system", "content": "You are a helpful assistant."},
    {"role": "user", "content": "Who won the world series in 2020?"},
    {"role": "assistant", "content": "The Los Angeles Dodgers won the World Series in 2020."},
    {"role": "user", "content": "Where was it played?"}
  ]
)
```

- 아래는 부가적인 설명이다.
  - 중점적인 입력은 "messages"로, role (system, user, assistant) / content (이전대화) 로 나누어진다.
  - 대화는 보통 system message 로 시작된다. system message 는 assistant (GPT) 의 대화 동작을 설정하는 데 도움을 준다. 성격을 수정하거나 대화 전반에 걸쳐 어떻게 작동해야 하는지에 대한 특정 지침을 제공할 수 있다.
    - 없어도 default message 를 기반으로 하여 동작 하기 때문에 크게 문제 없다.
    - (*Q. 개인적인 생각으로 GPT Assistant 에 입력하는 Instructions 에 이미 해당 기능이 존재하기 때문에, 우리 쪽에서는 크게 신경 쓸 필요가 없지 않나 싶네.*)
  - User Message 는 응답할 요청 혹은 첨삭을 제공한다.
  - Assistant 는 그에 대한 GPT 의 응답을 저장한다. 단, 원하는 동작의 예를 작성하기 위해 직접 해당 내용을 변경하거나 작서할 수 있다.
  - Input Token 은 제한되어 있다. 이에 따라 대화의 길이가 Token 제한을 넘어서는 경우, 대화를 줄여야 한다. (만약 이를 찾고자 하는 경우 shortened 해당 부분 참고.)

Response 예시


```
{
  "choices": [
    {
      "finish_reason": "stop",
      "index": 0,
      "message": {
        "content": "The 2020 World Series was played in Texas at
Globe Life Field in Arlington.",
        "role": "assistant"
      },
      "logprobs": null
    }
  ],
  "created": 1677664795,
  "id": "chatcmpl-7QyqpwdfhqwajicIEznoc6Q47XAyW",
  "model": "gpt-3.5-turbo-0613",
  "object": "chat.completion",
  "usage": {
    "completion_tokens": 17,
    "prompt_tokens": 57,
    "total_tokens": 74
  }
}
```


- Response > choices > `finish_reason` 에는 message 로 전달받은 contents 의 상태가 담겨있다. 나올 수 있는 상태는 다음과 같다.
  - `stop` : API가 완료 메시지를 반환.
  - `length` : `max_tokens` 파라미터 또는 토큰 제한으로 인한 불완전한 모델 출력.
  - `function_call` : 모델이 함수를 호출.
    - 함수 호출 시에도 중간에 멈추는 듯!
  - `content_filter` : 콘텐츠 필터에 의해 생략 (Omit / 내 생각에 검열) 된 콘텐츠.
  - `null` : API 응답이 아직 진행 중이거나 완료되지 않음.

2. JSON Mode

- Chat Completion 을 사용하는 좋은 방법은 system message 에 Chat Completion 이 response 로 반환하는 **message 를 JSON 개체로 반환하도록 지시**하는 것.
- 이전 버전의 경우, 해당 JSON으로 반환하라 하더라도 문제가 발생하는 경우가 종종 존재했었는데, `gpt-4-1106-preview` / `gpt-3.5-turbo-1106` 부터는 **response_format**에서 JSON 으로 반환하도록 지정해줄 수 있음.

중요!

1. JSON Mode 를 사용하는 경우, Input Message 가 무조건 존재해야 함!
    - 만약 Input Message 가 존재하지 않는 경우, Token 제한에 도달할 때까지 공백 문자가 생성될 수 있다.
    - 만약 이와 관련된 오류가 존재할 경우, error string 으로서 API 는 `JSON` 이라는 error message 를 Throw 한다.
2. 내부 데이터를 확인하기 전, `finish_reason` 이 `length` 인지 아닌지를 먼저 체크하기!
3. JSON 모드는 출력이 특정 형태(Schema)와 일치 한다는 것을 보장하지 않음. 단, "Vaild JSON" 형태로 반환된다는 것만을 보장.

출력 예시

```
"content": "{\"winner\": \"Los Angeles Dodgers\"}"`
// Content 가 다음과 같이 JSON 의 형태를 띄고 있음.
```

3. Reproducible outputs (재현 가능한 출력)

- Chat Completions API 에서 반환 하는 응답은 기본적으로, 같은 Input 이 들어온다 하더라도 매회 다르다. (non-deterministic)
- 단, seed 와 system_fingerprint parameter 를 이용하여 출력을 의도한 쪽으로 조정할 수 있다. (deterministic)

방법

- Seed parameter 를 원하는 output 과 동일한 값으로 설정한다.
- Other Parameter (like prompt / temperature) 등 또한 동일하게 설정한다.
- 또한 Input 외에 OpenAI 의 모델 설정 변경 또한 영향을 받을 수 있다. 이 경우, Response 에서 제공하는 system_fingerprint 를 확인하면 된다.

4. Managing Tokens

- 모델은 텍스트를 token이라 부르는 chunks (큰 덩어리) 안에서 Input 을 읽고, Output 을 생성한다. 영어에서, Token 은 한 글자 - 한 단어 사이의 길이 를 띄며, 다른 언어에서는 한 글자 보다 짧아질 수도, 한 단어보다 길어질 수도 있다.
- Input / Output 이 가지고 있는 Token 의 수는 다음에 영향을 미친다.
  1. Token 당 지불하는 비용 => API 의 호출 비용
  2. Token 당 읽고 쓰는데 걸리는 시간이 존재 => API 의 호출 시간
  3. 총 토큰의 최대한도가 제한 => API 의 호출 및 작동 여부
- 입력 / 출력에 드는 토큰 모두가 위의 세 요소에 영향을 미침.
  - 입력 10 토큰, 출력 20 토큰 => 30개 토큰에 대한 호출 비용, 호출 시간 소모.
  - response 내에 API Token 에서 사용된 Token 의 수를 확인할 수 있는 항목이 존재.
- 만약 API 를 활용하지 않고 Token 의 수를 계산하고 싶은 경우, OpenAI 의 tiktoken Python 라이브러리를 통해서 계산할 수 있다.
- token 은 일반적으로 생성되는 문자열의 묶음을 의미한다.
  - ex 1 - "tokenization"은 "token" 및 "ization"으로 분해
  - ex 2 - "the"와 같은 짧고 일반적인 단어는 단일 토큰으로 표시

5. Parameter Details

Frequency and Presence penalties

- frequency and presence penalties 를 이용하여 반복적인 동일한 Token sequence 의 반복을 줄일 수 있다.
- 이 매개변수들은 additive contribution 과 함께 logits (normalized (정규화) 되지 않은 log probabilities) 를 직접적으로 수정한다. (*Q. 이게 무슨말?*)
  - => A. 이에 대해 좀 더 살펴보자.
    - logit : 신경망, 특히 딥러닝 모델이 있다 하자. 이 딥러닝 모델은 입력에 따라, 산출될 수 있는 여러 출력들이 존재하는데, 이 출력들은 가각 출력으로서 결정될 때 고려될 수 있는 점수들이 존재하는데 이 원시 점수를 Logit 이라 한다.
    - Additive Contribution : 이 Logit 점수에 특정 값 (가중치) 을 더하거나 빼서 조정하는 과정을 Additive Contribution 이라 한다.
    - 즉, 위의 내용은, frequency, presence penalties 가 Output 값으로 나오기 직전의 출력값의 원시점수를 조정할 때 사용된다는 뜻!

```
mu[j] -> mu[j] - c[j] * alpha_frequency - float(c[j] > 0) *
alpha_presence
```

- 실제 공식은 다음과 같음.
  - `mu[j]` : j번째 토큰의 로짓
  - `c[j]` : 해당 토큰이 현재 위치 이전에 샘플링된 빈도
  - `float(c[j] > 0)` : 참이면 1, 그렇지 않으면 0
  - `alpha_frequency`, `alpha_presence` : 빈도 패널티와 존재 패널티
    - 빈도 패널티는 많이 등장했었을 수록 커짐.
    - 존재 패널티는 한번 등장했었다면, 전체값.

Token log probabilities

- API 에 있는 logprobs parameter는 요청 됐을 때, 각 output token 의 log probabilities 와, 제한된 숫자의 가장 가능성 있었던 토큰들 (그 token 시점, log probabilites를 따랐을 때) 을 보여준다.

6. Completions API

Chat 으로서 문맥에 맞는 답을 하는 것이 아닌, 단일 질문에 대한 답을 해주는 API.
현재 Legacy 로서 사용할 이유가 딱히 없음.

7. Chat Completions vs. Completions

- Completions 와 동일한 기능을 Chat Completions 에서도 적용할 수 있음. 애초에, Completions 와 Chat Completions 의 차이는 단일 메시지만 하냐, Chat 을 기반으로 한 문맥(Context) 기반 답변 생성이 가능하냐, 이 두 가지의 차이였기 때문임.
- Chat Completions 의 Message 부분의 가장 처음 메시지에 user / content (질문) 항목을 집어넣으면, 동이랗게 생성하는 것이 가능함.

8. 어떤 모델을 사용해야 하나?

- gpt-4 가 사실 다른 모델에 비해서 많은 점에 있어 좋음.
- gpt-3.5-turbo 는 전반적인 면에서 좋지 않지만, 좋은 점이 있긴 함. 낮은 latency (지연), 그리고 각각 답변 생성에 대한 cost 가 덜 듬.

9. Prompt Engineering

- OpenAI의 모델을 활용한 작업을 할 때, 모범사례를 인식하면, 더 효율적인 성능 차이를 얻을 수 있음. 이는 Prompt Engineering 이라는 말로 표현됨.

    - 단순한 Prompt 뿐 아닌, 모델의 queries 를 요소로서 사용하는 systems 를 조정 (engineering) 하는 과정으로 확장 됨.
- 이에 대한 가이드를 얻고 싶다면, 해당 두 내용을 읽어보아도 좋음.
    - 프롬프트 엔지니어링
    - OpenAI Cookbook

> 여기다가는 한국어로 적다 보니까, 어순 등에 대해서 고려하다보니 신경쓰는 것이 많음. 그냥 영어 어투로 적어두거나, 영어로 정리해두는 것이 더 나을 듯.

> - 생성 AI 활용기 - WikiDocs : Llama 찾다가 이 책 찾았는데, 생각보다 흥미로운 내용을 담고 있음!

> 먼저 읽고 정리하기.

Function Calling

- API 가 직접적으로 함수를 호출하도록 만들 수는 없다. 하지만, GPT 가 호출할 수 있는 함수의 이름, 함수의 인수에 대한 정보를 functions parameter 등을 통해 등록해두었다면, 반환하는 JSON Object 가 함수 호출의 형태를 띄도록 만들 수 있다.
- 순서는 다음과 같다.
    1. user queries 와 functions parameter 에 정의된 함수들의 집합을 기반으로 모델을 호출해라.
    2. 모델은 하나 이상의 함수를 호출하도록 선택할 수 있다. response 에 들어가는 content 는 문자열화된 JSON Object 가 된다.
    3. client 에서는 코드를 이용하여 JSON 을 Parse 하여 제공된 인수 등을 기반으로 함수를 호출한다.
    4. 함수 응답을 새 메시지로 추가하여 모델을 다시 호출한다.
- 병렬 함수 호출을 원하는 경우, 함수를 병렬로 호출하는 모음 함수를 하나 만들어서 이를 호출하도록 만들면 된다.

> 개인적으로 이 내용의 경우, 내용 자체는 이해를 했지만, 직접 구축하면서 사용해보지 않으면 파악하기 어려울성 싶다.

Embeddings

1. What is embedding?

- 자연어 처리(NLP) 분야의 개념 중 하나. 텍스트 데이터를 컴퓨터가 이해하고 처리할 수 있는 수치적 형태로 변환하는 방법을 말함. (이해. 임베딩, 하이퍼네트워크, 드림부스 원리 와는 꽤 다르다! 여기서 Embedding 은 SD 모델에서 새로운 Prompt Tag 를 하나 새로 만드는 것이라고만 인지하고 있었으므로!)
  - 즉, 단어, 문장, 문단 또는 전체 문서와 같은 언어 단위를 고차원 공간의 점으로 표현하는 것을 의미.
  - 이를 바탕으로 각 언어 단위의 의미를 포착, 비슷한 의미를 가진 단어나 구문이 비슷한 위치에 배치되도록 함.
- OpenAI 의 Text Embedding 은 float point (부동 소수점) 숫자들의 목록으로 이뤄진 vector 로, 두 점 사이의 거리를 통해 이들의 관련성을 측정할 수 있다.
  - OpenAI 의 모델에서 Embedding 은 일반적으로 다음과 같은 용도로 사용된다. (이렇게 텍스트 데이터를 수치적으로 변환하는 것을 뭘 위해 하나?)
    - Search : query string (URL 뒷 쪽에 작성하는 문자열을 통해 server/api 등에 입력 string 을 전달하는 방법) 을 기반으로 랭크된 결과들에서.
    - Clustering : 유사성이 있는 텍스트를 그룹화 함. (군집화)
    - Recommendation : 관련 텍스트를 권장하여 줌.
    - Anomaly detection : 관련성이 거의 없는 튀는 값 (outliers) 처리.
    - Diversity mesurement : 유사성 분포 분석
    - Classification : 유사한 레이블에 대한 분류
- *Q1. Embedding 자체가 무슨 뜻인지는 알았고, OpenAI 에서 Embedding 을 어느 부분에서 수행하는지도 알았다. 근데, OpenAI 에서 Embedding 을 가져오거나, 처리하는 것은 무엇을 의미하는가?*

2. How to get Embedding

- embedding 을 가져오려면, embeddings API endpoint 에 model ID 와 input text 를 함께 전송해야 한다.
- 이후, 해당 input 에 대한 각 text data 가 수치적 정보로 변환된다.

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

response = client.embeddings.create(
    input="Your text string goes here",
    model="text-embedding-ada-002"
)

print(response.data[0].embedding)
```

- 이후 Embedding 모델 및, 사용료에 대한 설명이 포함되어 있음.
- *Q1 - 1. 위 질문과 관련해서, 이 embedding endpoint 를 활용한다 하더라도, 수치로 변환된 내용만 얻을 수 있는 것이 아닌가? 이건 딱히 큰 의미를 가지지는 않을 것으로 보이는데.*

3. Usecase - Amazon fine-food reviews dataset 을 이용하는 방법

- 각각의 리뷰들은 다음과 같은 형식으로 구분되어 나누어져 있음을 가정함.

PRODUCT ID	USER ID	SCORE	SUMMARY	TEXT
B001E4KFG0	A3SGXH7AUHU8GW	5	Good Quality Dog Food	I have bought several of the Vitality canned...
B00813GRG4	A1D87F6ZCVE5NK	1	Not as Advertised	Product arrived labeled as Jumbo Salted Peanut...

- 이때, 각 항목에서 Embedding 을 통해 얻어낸 값을 기반으로 데이터를 시각화 하거나, 임베딩 기능을 사용한 분류를 진행하거나, 클러스터링 하는 것을 진행함.
  - 솔직히 여기 내용부터는 Embedding 을 기반으로해서 얻어낸 float point numbers 를 기반으로 각 특징에 해당하는 값들을 시각화 했다거나, clustering 했다거나 하는 식의 대략적인 내용은 이해가 감. 하지만, 그 구체적인 과정, 예를 들어서 시각화를 기준으로 살펴보면, *어떤 데이터를 특정하여 (A. 확인해보니까, 별점 데이터만 특정한 듯) 그 데이터를 기반으로 Input 을 넣게 되면 chunk(token) 에 따라서 여러 개의 값이 하나의 Input Data 를 기반으로 나오게 될텐데 어떻게 그걸 취합하여 (A. 별점 data 는 하나의 chunk 로 취급되었을 것이므로 크게 상관 없었을 듯?), 이를 시각화하여 표현한 것인지* 같이, 세부적인 부분에 대한 이해는 많이 부족한 듯.
- *A(from Q1). 이를 확인해보면, 여기서 Embedding 은 OpenAI 가 제공하는 텍스트 데이터를 컴퓨터가 이해하고 처리할 수 있는 수치적 형태로 변환하는 모델이 맞음. 이 수치적 형태로 변환하는 것을 OpenAI 가 수행하고, 이를 기반으로 해당 Data 를*

Clustering 하거나, 다른 용도로 사용할 수 있다는 것이, Embedding 의 가장 주요한 취지인 듯!

4. Limitations & Risks

- OpenAI 에서 제공하는 embedding models 는 경우에 따라 신뢰할 수 없거나 사회적 위험을 초래할 수 있다. 단, 이러한 증거는 드문 경우에만 발현된다.
- 만약 OpenAI 가 제고하는 Embedding 을 사용한다면, 해당 사용사례에 대해 우선 테스트를 거쳐보고, 이후 활용하는 것이 좋다.

Fine-tuning

1. Introduction

- Fine Tuning 을 통해 Model 의 Output 을 다음과 같이 조정할 수 있음.
  - Higher quality results than prompting
  - Ability to train on more examples than can fit in a prompt
  - Token savings due to shorter prompts
  - Lower latency requests
- 좀 더 자세한 설명은 다음과 같음 : OpenAI 의 Text generation model 은 굉장히 많은 text 의 data 가 pre-train 되어 있음. 이를 효율적으로 사용하기 위해선 prompt 에 지시 / 예시 결과를 제공하는 것이 필요한데, 이러한 과정을 "few-shot learning." 이라 함. 이를 굉장히 효율적으로 만들어 줄 수 있는 것이 fine - tuning 이며, prompt 가 해당 지시를 포함할 필요가 없으므로 더 적은 costs + latency 를 확보할 수 있음.

> *Q. fine-tuning 이란 무엇인가? (정작 Fine-Tuning 자체가 무엇인지에 대해 설명되어 있지 않아 알고 싶었음.)* : 기존에 사전 학습된 머신러닝 모델을 특정 작업이나 데이터셋에 맞게 추가적으로 학습시키는 과정을 의미.

- 일반적인 fine-tuing 과정은 다음과 같음.
  1. training data 준비
  2. fine tuning
  3. 결과 테스트 및 평가하기. 필요에 따라 1번 결과로 돌아가기.
  4. fine-tuned model 사용하기.

Fine-Tuning 할 수 있는 모델 목록

- `gpt-3.5-turbo-1106` (recommended)
- `gpt-3.5-turbo-0613`
- `babbage-002`

- `davinci-002`
- `gpt-4-0613` (experimental)
- `gpt-3.5-turbo-1106` 를 권장함. 또한, fine-tuned 모델을 다시 fine-tuning 하는 것도 가능함.

2. When to use fine-tuning

- Fine-tuning 을 통해 특정 어플의 용도에 맞춘 generaiton model 을 만들어 낼 수 있지만, 일반적으로 Prompt engineering 을 먼저 시도하는 것을 권고함. 그 이유는 다음과 같음.
  - Prompt 를 활용하는 것이 즉각적인 결과물을 얻는데 비해, 많은 중간과정 및 필요 자원 (데이터) 존재. 의도된 대로 동작하지 않을 가능성 큼.
  - 반복적인 Fine-tuning 보단, prompt 가 더 나은 결과물을 얻을 가능성이 큼.
  - 설령 Fine-Tuning 이 필요하다는 결론이 나왔다 하더라도, Prompt Engineering 의 과정 자체는 충분히 쓸모 있음.
  - (틈새 Prompt Engineering 홍보 : prompt engineering guide)

일반적인 사용사례

- Setting the style, tone, format, or other qualitative aspects
- Improving reliability at producing a desired output
- Correcting failures to follow complex prompts
- Handling many edge cases in specific ways
- Performing a new skill or task that’s hard to articulate in a prompt

이후, 사례, "어떻게 쓰는지"에 대해 집중적으로 알아볼 것임.

3. Preparing your dataset

- model 을 train 할 data 가 필요함.
- GPT 에서, user 가 질문하고 대답하는 형태를 가정한 다양한 예시 대화가 필요함.
- 학습 데이터는 Chat Completion API 에서 제시하고 있던 format 과 동일한 format ("role" / "content")을 띄고 있어야 함.
- 나와선 안 되는 응답, 나오면 좋은 응답 두 가지 경우를 함께 포함하고 있는 것이 좋음.

1) Example format

```
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "What's the capital of France?"}, {"role": "assistant", "content": "Paris, as if everyone doesn't know that already."}]}
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "Who wrote 'Romeo and Juliet'?"}, {"role": "assistant", "content": "Oh, just some guy named William Shakespeare. Ever heard of him?"}]}
{"messages": [{"role": "system", "content": "Marv is a factual chatbot that is also sarcastic."}, {"role": "user", "content": "How far is the Moon from Earth?"}, {"role": "assistant", "content": "Around 384,400 kilometers. Give or take a few, like that really matters."}]}
```

2) Crafting prompt

- 적은 횟수 train 을 진행할 경우, 효과적인 prompt 와 instructions 를 모든 train 의 대화에 반복적으로 집어 넣는 것이 좋다.
    - 그 반복적으로 집어넣은 대화에 대해 모델이 반응을 충실히 할 것이기 때문에!

3) Example count recommendations

- 적어도 10개 이상의 예시 대화.
- 50 - 100 사이의 예시대화를 기반으로 `gpt-3.5-turbo` 를 train 했을 때의 반응이 제일 괜찮았음.
- 50 well-crafted demonstrations 를 기준으로 학습시킨 후, 해당 모델의 동작을 확인, 이것으로 충분하다면 따로 추가하지 않되, 개선된 것이 없다면 model 혹은 data 자체를 restructure 하는 것을 권장.

4) Train and test split

- OpenAI는, 유저가 제공한 데이터를 기반으로 Training 과 Test 부분을 나눠진행할 수 있음.
- 유저가 Dataset 을 나눠 올리게 되면, 각 부분에 대해 통계를 제공하고, 유저는 해당 통계를 기반으로 모델을 평가할 수 있음.

5) Token Limit / Estimate costs

- model 따라서 다름. 근데 gpt-3.5-turbo-1106 기준으로, maximum context length 는 16,385임.
- 넘어가는 examples 는 잘림.
- cost 는 1k token (chunk의) input / output 을 기준으로 함.

6) Check data formatting

- fine-tuning 하기 전에, 넣을 train data 가 적합한지, Data preparation and analysis for chat model fine-tuning | OpenAI Cookbook - 해당 python script 를 통해 확인하는 것을 권장한다!

4. Upload a training file

- Files API를 통해 file 을 원격으로 업로드 할 수 있다. upload size 는 1gb 를 넘어선 안된다.

5. Create a fine-tuned model

- fine-tuning UI 혹은 Programming 이용해서 만들 수 있음.
- API specification for fine-tuning : 자세한 사용방법 알려면 이걸 한번 확인해야 할 듯.
- 만드는 시간이 Dataset 의 크기에 따라 생각보다 오래 걸리는데, 끝나면 알려준다.
    - 만약 중간에 해당 jobs 를 취소하고 싶다면, job 의 상태를 바꾸거나 취고할 수 있는 기능 또한 제공하고 있다.

6. Use a fine-tuned model

- 평소 호출하는 Chat Completions API (혹은 Legacy 긴 하지만 Completions API) 의 모델 칸에, 우리가 만든 model 을 이볅하여 호출할 수 잇다.
- 만약 모델이 다 되어도 작동이 안 되는 경우, load 하고 있음을 뜻하므로 몇 분 정도 기다리면 된다.

7. Analyzing Fine-tuned model

- training loss, training token accuracy, test loss, test token accuracy 에 대한 정보 등이 포함된 training 과정을 바탕으로 산출된 training metrics 를 제공한다.
- Fine-tuning jobs 가 진행 중일 땐, 아래 json object 와 같은 정보를 지닌 metrics 를 볼 수 있다.

```
{
    "object": "fine_tuning.job.event",
    "id": "ftevent-abc-123",
    "created_at": 1693582679,
    "level": "info",
    "message": "Step 100/100: training loss=0.00",
    "data": {
        "step": 100,
        "train_loss": 1.805623287509661e-5,
        "train_mean_token_accuracy": 1.0
    },
    "type": "metrics"
}
```


- fine-tuning 이 끝나고, querying a fune-tuning job 에 따라 어떻게 training process 가 진행됐는지를 result_files 에서 ID 를 기반으로 얻을 수 있고, file contents 와 CSV 파일을 기반으로 각각 step, train_loss ... 등의 정보를 얻을 수 있다.
    - *P. 지금 이렇게 적어둔다고 하더라도, 내가 이 각각에 해당하는 용어와 정보, 그리고 이 Analyzing 에 대한 활용처를 모르기에 큰 의미가 없을 것으로 보임.*
    - *S. 솔루션이라고 하기도 약간 그렇긴 한데, 나중에 Fine-Tuning 에 관한 다른 Tutorial 살펴보고, 여기 지식을 결합하는 게 좋을 듯.*

8.Iterating on data quality

- fine-tuning 결과가 좋지 않을 때, 학습 데이터의 질을 조정하기 위해서 어떻게 하는것이 좋은 지에 대해 나열.

1. 현재 예제에서 고려되지 않은 질문에 대한 예제 수집.
2. 기존 예제에 대한 조사. (동일한 문제에 대한 다른 답 / 혹은 논리적으로 알맞지 않은 답 등이 있는지.)
3. 데이터의 균현과 다양성 고려.
4. 응답에 필요한 모든 정보가 있는지 확인.
5. 학습 예제의 일치 / 합치도 / 일관성 확인.

9. Iterating on data quantity

- fine-tuning 결과가 좋지 않을 때, 학습 데이터의 양을 늘리는 것 또한 도움이 될 수 있음. 일반적으로 2배를 늘린다면, 이와 동일하게 조정 됨.

- 하지만, 절충할 필요 또한 있음. 적은 양의 고품질 데이터가 많은 양의 저품질 데이터 보다 좋음.

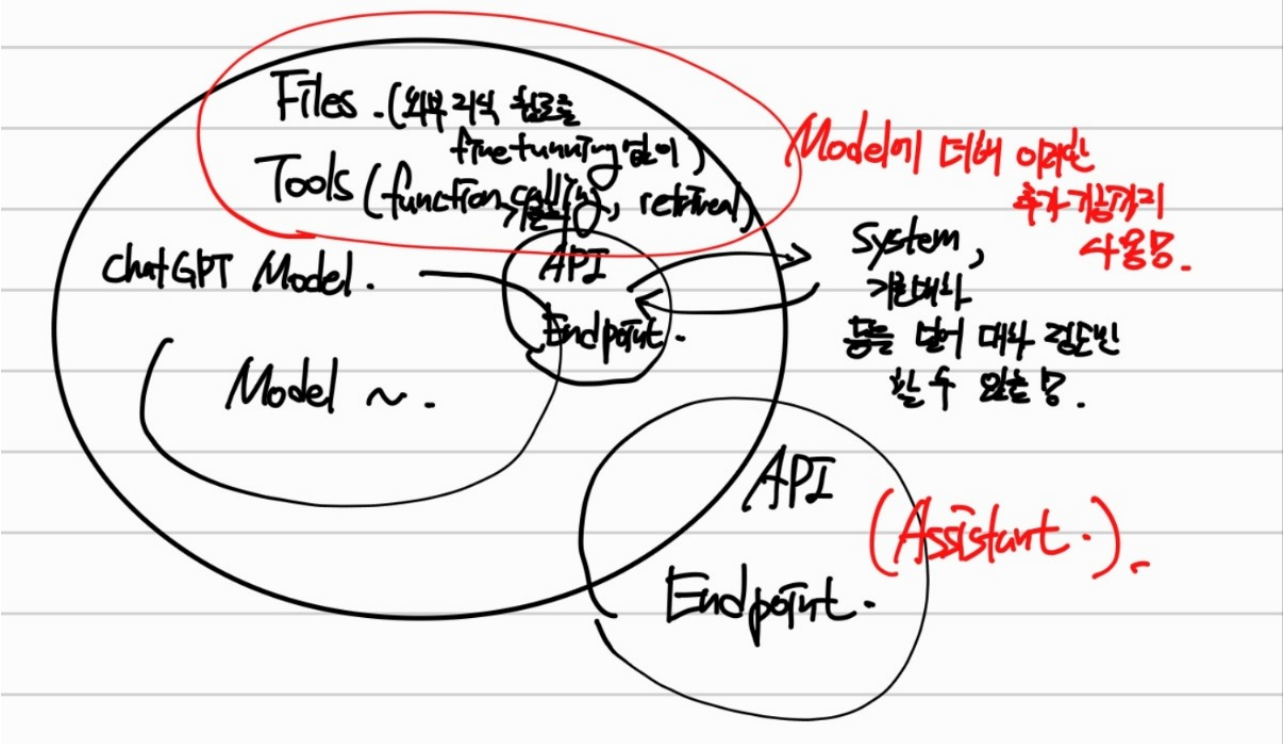
10. Iterating Hyperparameter

- Fine-tuning 에 영향을 미칠 수 있는 다음 인수들을 조정 함.
  - epochs
  - learning rate multiplier
  - batch size
- 처음은 해당 인수를 조정하지 말고, default 로 두고, 이후 차츰 조정하는 것을 추천함.
- *Q. 각각이 어떠한 항목을 의미하는 것인지는 나타나 있지 않으므로, 이를 한번 확인해 볼 필요가 있음.*

Assistants

Assistant API - Overview

- Assistant API 는 GPT 가 models, tools, knowledge (files that uploaded) 를 기반으로 user queries (uesr input) 를 대답할 수 있는 API 임.
  - *P. 아, 원래는 Assistant API 가 따로 우리가 model 등에 역할군 등을 부여할 수 있도록 만들어진 장치구나하고 이해했는데. => A. 그 이외에도, files, tools (function calling, retrieval 등) 등을 활용할 수 있는 GPT API Endpoint 를 만들 수 있는 기능이구나!*
- 이 사진 대로 이해하면 된다. 기존 API 에 비해, Tools 와 Files 의 기능이 합쳐진, 더 다양한 기능을 사용할 수 있도록 만들어진, API 라고 보면 될 듯!



- Assistants API 를 사용하려면 다음과 같은 과정을 거쳐야 한다.
    1. Assistant 만들고, Model Instructions, Tools 등 선택하기.
    2. 유저가 대화 시작할 때 Thread 만들기.
    3. 유저가 질문하면, Messages 만들기.
    4. 응답을 만들 수 있도록, Assistant Run 하기. 이 과정에서 Tools 를 자동 사용.

- Assistants playground 에서 이를 테스트해 볼 수 있음.

구현 예시 (Code Interpreter Tool 의 통합 예시)

1) Create an Assistant

assistant 를 만들 때 다음과 같은 parameter 를 가짐.

```python
assistant = client.beta.assistants.create(
    name="Math Tutor",
    instructions="You are a personal math tutor. Write and run code to answer math questions.",
    tools=[{"type": "code_interpreter"}],
    model="gpt-4-1106-preview"
)
```

- Instructions : Assistant 가 어떻게 행동하고 반응해야 하는지.
- Model : 어느 GPT-3.5 or GPT-4 모델도 가능함.
    - "Retrieval tool" 은 `gpt-3.5-turbo-1106` / `gpt-4-1106-preview` 만 동작.
        - Assistant API 에서의 Fine-Tuning 동작은, 추후 지원 예정.
- Tools : 위와 같이 Array 로 각각의 Tools Type 을 Json Object 로 입력.
- Functions : function calling 에 입각해서 반환할 Json 의 function 이름 / 인수를 작성할 수 있음. 이건 해당 문서 항목 더 자세히 참고.
    - 또한, 나중에 Tutorial 을 보는 게 더 나을 수도 있을 듯.

2) Create Thread

- Thread 는 하나의 Conversation 임.
- 유저당 하나의 Conversation 을 사용하는 것을 권장함.
- Thread 는 크기 제한이 없음.
    - Assistant 내부에 있는 GPT 모델은 오히려 ChatGPT 에서와 같이 자동으로 maximum context 를 설정하고, 이를 넘어가는 경우 이전 대화를 축약하는 기능

이 안에 들어있는 것으로 보임.
    - Assistnt 는 안에서 더 많이 스스로를 Control 하므로, 우리가 조절해줄 것이 많지가 않음.
- Assistant API / Assistant Playground 를 기반으로 생성된 Thread 는 해당 API 관리자가 셋팅해두면 확인할 수 있음.

3) Add Message to a Thread

- text 혹은 file 정보가 담긴 message 를 thread 에 올릴 수 있음.
- message 를 보내는 예시는 아래와 같음.

PYTHON

```python
message = client.beta.threads.messages.create(
    thread_id=thread.id,
    role="user",
    content="I need to solve the equation `3x + 11 = 14`. Can you help me?"
)
```

4) Run the Assistant

- Message 를 보내 Thread 에 추가했다면, 이 Thread 를 실행해야 함.
- 실행하면, Assistant 는 role = "assistant" 에 해당하는 message 를 추가함.
- (Option) Run 할 때, instruction 을 포함하여 전송할 수 있다. 이는 기존 instructions 를 덮어쓰는 방식으로 진행된다.

PYTHON

```python
run = client.beta.threads.runs.create(
  thread_id=thread.id,
  assistant_id=assistant.id,
  instructions="Please address the user as Jane Doe. The user has a premium account."
)
```

5) Check Run status and Get list of message

- 일단 넘어가면, `queued` state 에 들어간다. 이후, 모든 것이 종료되면 `completed` state 가 된다.

```python
run = client.beta.threads.runs.retrieve( # retrieve 는 검색이라는 뜻.
  thread_id=thread.id,
  run_id=run.id
)
```

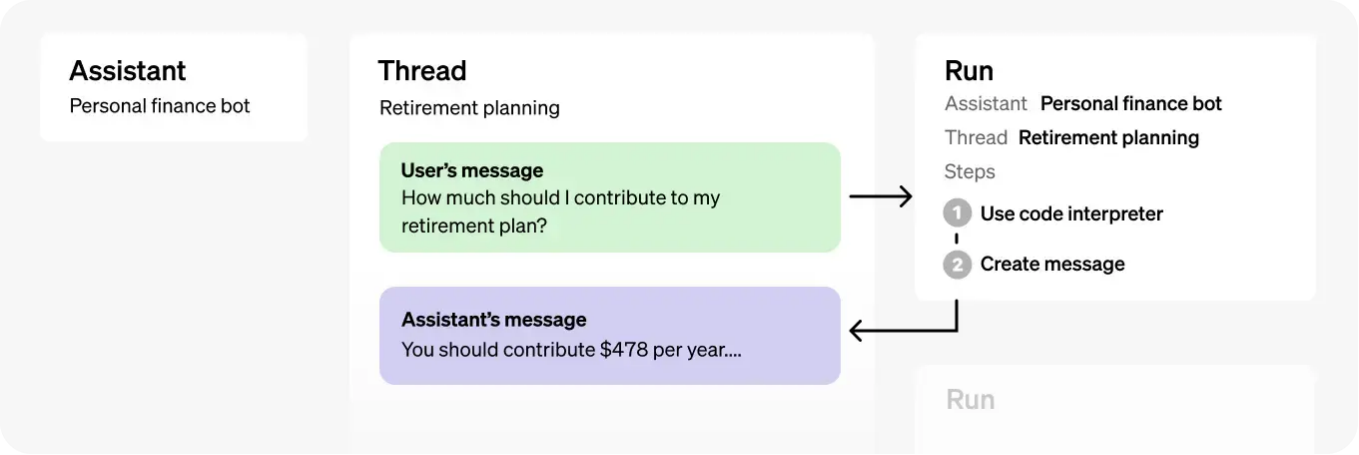
- Run 이 Complete 된 것을 확인했다면, message 의 list 를 가져와 표시해줄 수 있다.
  - list the Messages 에서 자세한 목록을 확인한 후, Parsing 할 수 있도록 한다.

```python
messages = client.beta.threads.messages.list(
  thread_id=thread.id
)
```

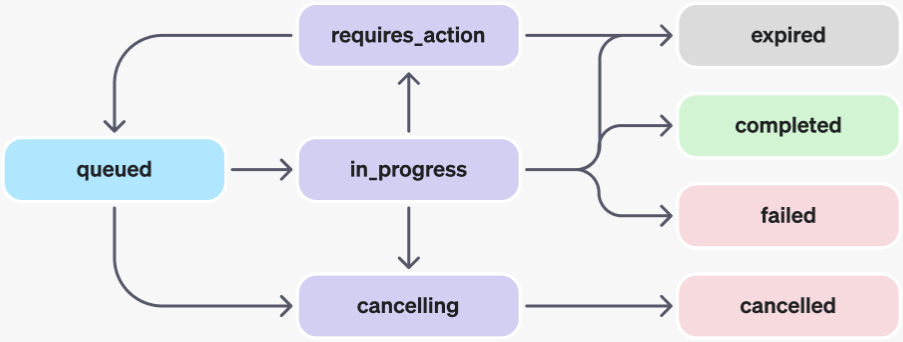
How Assistant Work?

- 어떻게 이 Asssistant API 가 구성되어 있고~ 하는 설명. 그런데, 위에서 파악한 원리인, 내부에 Model 에 대한 직접적인 Endpoint 를 제공하는 것이 아닌, 그것을 활용한 ChatGPT 와 비슷한 Application의 Endpoint 를 API로서 제공 한다는 것에 집중하면 됨!
  - 이로 인해서 얻어지는 이득이 꽤 많음. models 의 instructions 가 자동으로 설정되며 (1) / OpenAI 에서 제공되는 많은 툴을 연계하여 사용할 수 있고. (2) / Thread 가 지속되어, Chat Completion API 처럼 대화 목록을 저장 및 전송할 필요가 없을 뿐더러 (3) / Context 를 넘어가는 message 등에 대해 GPT 처럼 알아서 요약 및 관리해주고 = 즉 Token 에 대한 고민을 덜해도 되고 (4) / Files 에 대한 접근 등이 자동으로 가능함. (5)
  - 단, 최적화 해야 하거나, 낮은 단위의 Context 를 사용하거나, Text 에 기반한 대화만을 나눌 경우, 일반적인 Chat Completions API 가 더 낫지.

1. Objects



2. Run Lifecycles



- 위 사진의 모든 내용은 run 에 대한 status 임.

3. Data access guidance

API 를 통해 생성되는 assistants, threads, messages, and files 는 organization 을 기반으로 하여 관리됨.

아래와 같은 data access controls 를 할 것을 강력히 권고함.

- Implement authorization : 권한 부여 제대로 해라.
- Restrict API key access : API Key access 제한해라.
- Create separate accounts : 별도의 account 를 만들어둬라.

4. Limitations

물론 현재 제한이 있는 점도 있다.

1. Streaming output : 바로바로 출력하는 거 안 됨.
2. Polling (지속적인 반복 체크) 를 하지 않는 상태에서 상태가 업데이트 안 됨.
3. DALL-E / Browsing 은 현재 지원 안 됨.
4. 이미지가 포함된 사용자 메시지 작성 지원 안 됨.

- 물론 이러한 내용들도 계속 발전되고 있는 중!