

ChatGPTプラグイン

ChatGPTをサードパーティのWEBアプリケーションのAPIに接続するためのプラグインであり、GPT機能を外部の知識や計算能力と合わせて利用することが可能になる

- リアルタイム情報取得: スポーツスコア、株価、最新ニュース
- 知識ベース情報の取得: 会社のドキュメント、個人的なメモなど

*ユーザーに変わって行動を実行: フライトの予約、食べ物の注文など

ChatGPT plugins <https://openai.com/blog/chatgpt-plugins>

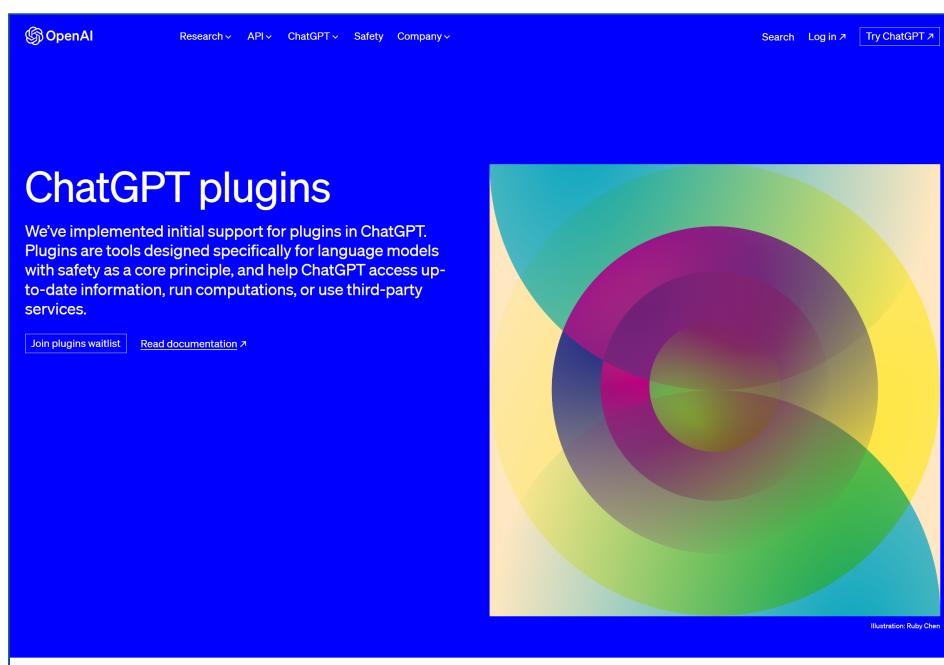


図7.1. 出所:OpenAIプラグイン公式Webサイト |OpenAIのプラグインホームページ

ChatGPT pluginsリスト

Plugins名	機能
AskYourPDF	PDFの要約と対話
Expedia	旅行・宿泊・交通・観光の計画やリアルタイム情報入手
FiscalNote	世界の法制度や規制、訴訟関連など、公的機関のリアルタイム情報入手
Instacart	お気に入り地元の食料品店に注文
KAYAK	フライト、滞在場所、レンタカー、予算内で行けるおすすめ場所の情報入手

Plugins名	機能
Llarna	オンラインショップの価格の比較情報入手
Milo	子育てサポートのためのTODOを管理
OpenTable	レストランのおすすめ情報入手
Shopify	ブランド品情報入手
Speak	任意言語を学ぶ
Tabelog	飲食店の情報入手
WebPilot	Web検索による情報入手
Wolfram	計算、数学、キュレーションされた知識、リアルタイム情報入手
Zapier	Googleスプレッドシート、Trello、Gmail、HubSpot、Salesforce5,000以上のアプリと対話

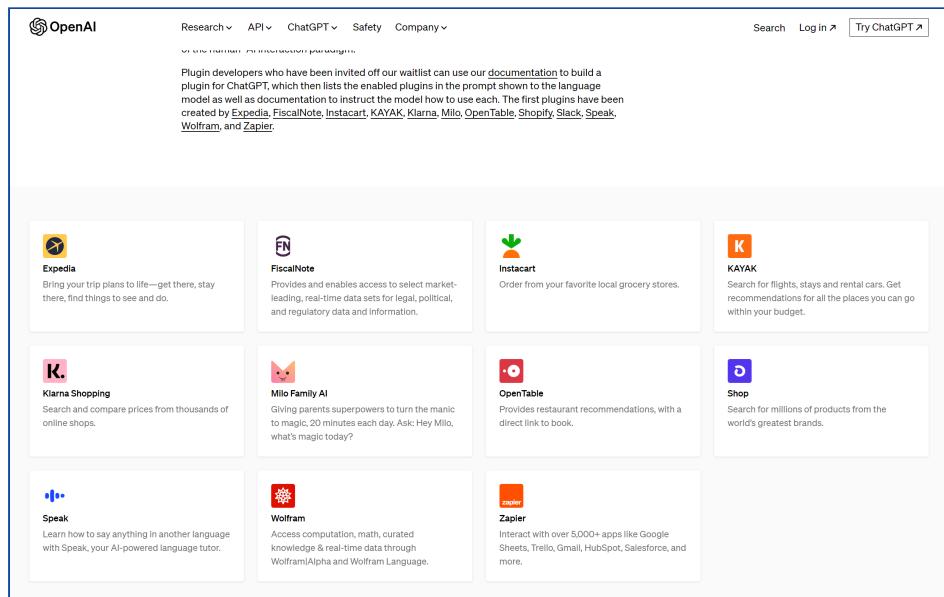


図7.2. 出所:OpenAIプラグイン公式Webサイト |ChatGPT有料プランにて提供されているプラグイン一覧

ChatGPTプラグインの使い方

- ChatGPTプラスプラン加入
- ウエイティングリスト登録
- ChatGPT PlusのGPT-4 -> Pluginsを選択 (Default/Bing/Pluginsという3オプションがある)
- Plugin storeをクリック
- 必要なプラグインアプリのアイコンを選択

- インストール
- プラグインの機能：

ChatGPTが入力質問内容に対しインストールされたプラグインを使用して回答を出力してくれる

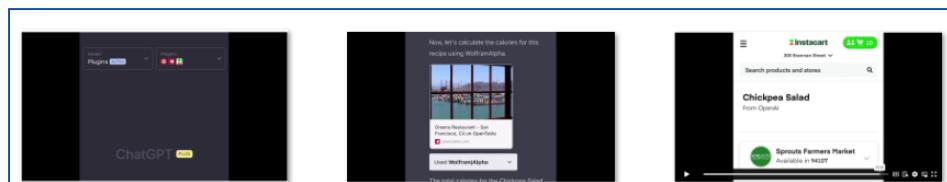


図7.3. 出所:OpenAIプラグイン公式Webサイト |ChatGPTのプラグイン一機能による出力例

ChatGPT(β版) プラグインの作り方

Chat Plugins Beta公式サイト

<https://platform.openai.com/docs/plugins/introduction>

プラグインを構築するエンドツーエンドフロー

1. マニフェストファイルを作成し、`yourdomain.com/.well-known/ai-plugin.json`にホストする。

- このファイルには、プラグインに関するメタデータ（名前、ロゴなど）、必要な認証の詳細（認証の種類、OAuthのURLなど）、公開したいエンドポイントのOpenAPI仕様が含まれている。
- 最初は1つまたは2つのエンドポイントのみを公開し、テキストの長さを最小限に抑えるために少ないパラメータを指定する。

2. ChatGPT UIでプラグインを登録

- トップのドロップダウンメニューからプラグインモデルを選択し、「プラグイン」、「プラグインストア」、最終的に「独自のプラグインを開発」を選択する。
- 認証が必要な場合、OAuth 2のclient_idとclient_secret、またはAPIキーを取得する。

3. ユーザーがプラグインをアクティブ化

- ユーザーはChatGPT UIでプラグインを手動でアクティブ化する必要がある。
(ChatGPTはプラグインをデフォルトで使用しない。)
- OAuthが必要な場合、ユーザーはOAuthを経由してプラグインにリダイレクトされ、サインインする必要がある。

4. ユーザーが会話を開始

- OpenAIはプラグインのコンパクトな説明をChatGPTにメッセージとして挿入し、エンドユーザーには見えないようになっている。
- ユーザーが関連する質問をすると、モデルはAPIコールを呼び出すことを選択する。POSTリクエストの場合、開発者は破壊的なアクションを回避するためのユーザー確認フローを構築する必要がある。
- モデルはAPIコールの結果をユーザーへの応答に組み込み、APIコールから返されたリンクを応答に含める。これらはリッチプレビューとして表示される（OpenGraphプロトコルを使用し、site_name、title、description、image、urlフィールドを抽出）。
- モデルはAPIからのデータをmarkdownでフォーマットでき、ChatGPT UIはmarkdownを自動的にレンダリングする。

カリフォルニア州での会話プラグインヘッダー例{"openai-subdivision-1-iso-code": "US-CA"}

プラグインの基本

- プラグインの構築を開始する

<https://platform.openai.com/docs/plugins/getting-started>

- サンプルプラグインを探索する

<https://platform.openai.com/docs/plugins/examples>

- プラグインを本番環境で展開するための重要なステップを読む

<https://platform.openai.com/docs/plugins/production/rate-limits>

- プラグインのレビュープロセスについて学ぶ

<https://platform.openai.com/docs/plugins/review>

- OpenAPI Initiative

<https://www.openapis.org>

<https://spec.openapis.org/oas/v3.1.0>

OpenAPIはWebAPI設計とドキュメント化を行うための仕様である。

APIのリソース、エンドポイント、パラメーター、ヘッダー、認証方法などを定義できる。

注意! OpenAI APIとOpenAPIは別物である。

ChatGPT Retrieval Plugin

- ローカルPCでOpenAI公式のChatGPTプラグイン実装「ChatGPT retrieval Plugin」を起動し、LangChainから操作する方法
- 情報公開を許可した個人や団体の独自データを使って質問応答することができる

プラグインの概要

openai/chatgpt-retrieval-plugin

<https://github.com/openai/chatgpt-retrieval-plugin>

ベクトルデータベースの準備

ドキュメントの情報保持と検索にベクトルデータベースを使用

データベースリスト

- pinecone

<https://www.pinecone.io>

- weaviate
- zilliz
- milvus
- qdrant
- redis

ベクトルデータベース作成

ベクトルデータベース作成画面に以下の内容を入力すれば、インデックス作成ができる。

データベースのインデックス名: test

Dimension: 1536

Metric: euclidean

Pod Type: starter (以外は有料版)

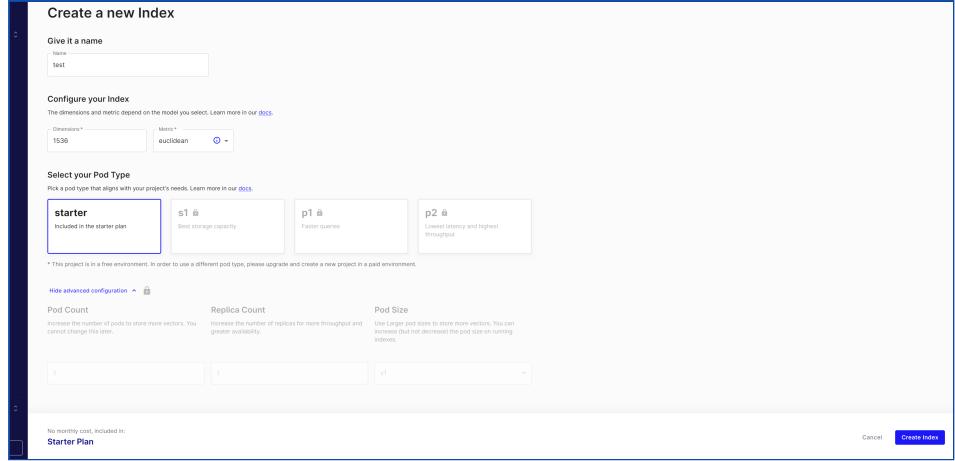


図7.3. 出所:Pinecone公式Webサイト |ベクトルデータベース作成画面

PineCone API キー取得

PineConeアカウント登録すれば、APIキーが取得できる。



図7.5. 出所:Pinecone公式Webサイト |PineConeAPIキーの取得画面

JWTトークンの準備

JWTトークン (JSON Web Token)は、APIへのリクエストを認証するための「シークレットトークン」である。

トークン公式サイト

<https://jwt.io/>

このサイトの自動生成トークンを利用する

Webアプリケーションの起動

```
In [ ]: #アプリ起動順番
#Anaconda
#Python3.10 Env
#プラグインのダウンロード/レポジトリークローン
# python -m venv myenv
# myenv\Scripts\activate
# pip install some-library
# python your_script.py

#プラグインのダウンロード/レポジトリークローン
```

```
git clone https://github.com/swagger-api/swagger-ui  
git clone https://github.com/tiangolo/fastapi  
git clone https://github.com/openai/chatgpt-retrieval-plugin.git  
git clone https://github.com/python-poetry/poetry.git
```

```
In [ ]: $cd chatgpt-retrieval-plugin  
$pip install poetry  
$poetry env use python3.10  
$poetry install
```

環境変数の準備

ローカルPCのシステム環境変数の設定をする必要がある。

環境変数	設定する値例
DATASTORE	pinecone
BEARER_TOKEN	JWTトークン
OPENAI_API_KEY	OpenAI APIキー
PINECONE_API_KEY	PineconeのAPI
PINECONE_ENVIRONMENT	Pineconeの環境
PINECONE_INDEX	Pineconeのインデックス名

```
In [ ]: # main.pyファイルを作成し、fastapiのサーバー接続を確認  
  
from typing import Union  
  
from fastapi import FastAPI  
  
app = FastAPI()  
  
@app.get("/")  
def read_root():  
    return {"Hello": "World"}  
  
@app.get("/items/{item_id}")  
def read_item(item_id: int, q: Union[str, None] = None):  
    return {"item_id": item_id, "q": q}
```

```
In [ ]: $poetry run start

# $ uvicorn main:app --reloadを実行すると下記のようにurlが表示される

# INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to qui
# INFO:     Started reloader process [28720]
# INFO:     Started server process [28722]
# INFO:     Waiting for application startup.
# INFO:     Application startup complete.
```

```
In [ ]: from enum import Enum

from fastapi import FastAPI

class ModelName(str, Enum):
    chapter1 = "第1章"
    chapter2 = "第2章"
    chapter3 = "第3章"

app = FastAPI()

@app.get("/models/{model_name}")
async def get_model(model_name: ModelName):
    if model_name is ModelName.chapter1:
        return {"model_name": model_name, "message": "データフロント 夜の煌びやかさ"}

    if model_name.value == "第2章":
        return {"model_name": model_name, "message": "ウルフ・コーポレーションの豊富な経験と技術で開発された"}

    return {"model_name": model_name, "message": "裏切りと再会 バー「グランマズ」での再会が物語の転機となる"}
```

```
In [ ]: # $ uvicorn main:app --reloadを実行すると下記のようにurlが表示される

# (env) C:\Users\user\env>cd C:\Users\user\env\fastapi-master\
# (env) C:\Users\user\env\fastapi-master>python main.py
# (env) C:\Users\user\env\fastapi-master>uvicorn main:app --reload

# INFO:     Uvicorn running on http://127.0.0.1:8000 (Press CTRL+C to qui
# INFO:     Started reloader process [000000]
# INFO:     Started server process [00000]
# INFO:     Waiting for application startup.
# INFO:     Application startup complete.
```

WebアプリケーションSwagger UIへアクセス

- <http://127.0.0.1:8000/docs>

*Swagger UIのAuthorizeボタンを押す（操作に必要なhtmlコードを用意）



The screenshot shows the MDASH-shinshu API documentation interface. A specific endpoint, 'default /models/{model_name} Get Model', is selected. In the 'Parameters' section, 'model_name' is set to '第1章'. Below it, there's an 'Execute' button and a 'Clear' button. The 'Responses' section shows a successful 200 status code response. The response body is a JSON object:

```
{
  "model_name": "第1章",
  "message": "データフロント 夜の煌びやかなネオ東京。高層ビルが連なり、ネオンが街を彩る。",
  "content": "その消費を減らすためにデータカウリヤに身を任せていた。ある日、ミコは重要なデータを運ぶ仕務を受ける。そのデータは、巨大企業「ウルフ・コーポレーション」による市民への無理な配分を監視するためのもの。彼女はデータを受け取り、目的地に向かうことを…"
}
```

Below the response body, there are sections for 'Response headers' (Content-Length: 880, Content-Type: application/json, Date: Mon, 01 Nov 2023 08:05:36 GMT, Server: unicorn) and 'Links' (No links).

図7.6. 出所:MDASH-shinshu github公式サイト |実行後に表示されるResponse画面例

インデックスの初期データの追加

```
In [ ]: #このファイルをアップロードし例のようにデータを追加
C:\user\user\chatgpt-retrieval-plugin\scripts\process_json\example.json

#アップロードファイルの中身の例

[
  {
    "id": "1",
    "text": "第1章:データフロント 夜の煌びやかなネオ東京。高層ビルが連なり、ネオンが街を彩る。",
    "source": "file",
    "source_id": "https://.json",
    "url": "https://",
    "created_at": "2021-01-01T12:00:00Z",
    "author": "著者名"
  },
  {
    "id": "1",
    "text": "第2章:ウルフ・コーポレーションの罠 ミコは、目的地にあるバー「グランマズ・ハウス」へ",
    "source": "file",
    "source_id": "https://.json",
    "url": "https://",
    "created_at": "2021-01-01T12:00:00Z",
    "author": "著者名"
  },
  {
    "id": "1",
    "text": "第3章:ミコの決意 ミコは、ウルフ・コーポレーションの本拠地である高層ビルに向かう。",
    "source": "file",
    "source_id": "https://.json",
    "url": "https://",
    "created_at": "2021-01-01T12:00:00Z",
    "author": "著者名"
  }
]
```

```
        "id": "1",
        "text": "第3章:裏切りと再会 バー「グランマズ・ハウス」で、ミコはデータの受け取り人・リョウを行方不明にしてしまった父の消息を聞きたい",
        "source": "file",
        "source_id": "https://.json",
        "url": "https://",
        "created_at": "2021-01-01T12:00:00Z",
        "author": "著者名"
    }
]
```

Webアプリケーションの動作確認

- <http://127.0.0.1:8000/docs%E3%81%AB%E3%82%A2%E3%82%AF%E3%82%BB%E3%82%BF>

*Swagger UIのAuthorizeボタンを押す（操作に必要なhtmlコードを用意）

*ValueにJWTトークンを入力しAuthorizeを押す

*「/query」を開き、「Try it out」ボタンを押す

*Request bodyに下記のパラメーターを入力する

```
{
    "queries": [
        {
            "query": "ミコの幼馴染の名前は?",
            "top_k": 2
        }
    ]
}
```

- Executeを押す

- Responseの確認

In []:

```
{ "queries": [
    {
        "query": "ミコの幼馴染の名前は?",
        "top_k": 2
    }
]}
```

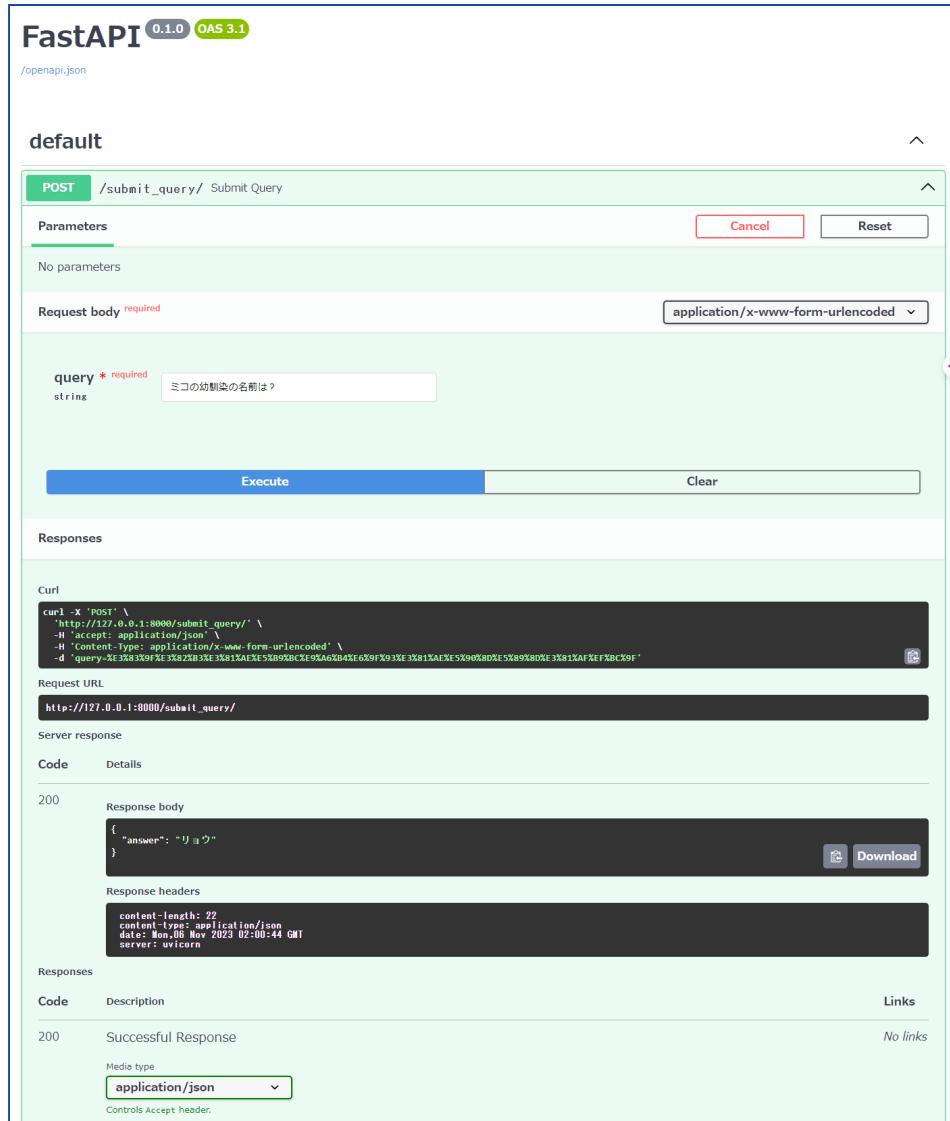


図7.6. 出所:MDASH-shinshu github公式サイト |実行後に表示されるResponse画面例

LangChainからの操作確認

```
In [ ]: !git clone git@github.com:openai/retrieval-app.git
```

```
In [ ]: cd /path/to/retrieval-app
```

```
In [ ]: !pip install poetry
```

```
In [ ]: !poetry env use python3.10
```

```
In [ ]: !poetry install
```

環境変数の設定

BEARER_TOKEN: <https://jwt.io/>

OPENAI_API_KEY: <https://platform.openai.com/account/api-keys>

PC DATASTORE: pinecone

PINECONE_API_KEY: <https://app.pinecone.io/>

PINECONE_ENVIRONMENT: us-east1-gcp, us-west1-aws, gcp-starter, etc.

PINECONE_INDEX: test

```
In [ ]: !pip install kaleido
!pip install python-multipart
!apt-get update
!apt-get install python3.10-venv
!pip install poetry
!pip install fastapi
!pip install uvicorn
!pip install "uvicorn[standard]"
!pip install lida
!pip install FastAPI -q
!pip install uvicorn -q
!pip install fastapi
!pip install requests
!pip install -q transformers
```

```
In [ ]: !poetry run start

# INFO:     Uvicorn running on http://0.0.0.0:8000
# INFO:     Application startup complete.
```

```
In [ ]: !pip install -qU datasets pandas tqdm
```

```
In [ ]: from datasets import load_dataset

data = load_dataset("squad", split="train")
data
```

```
In [ ]: data = data.to_pandas()
data.head()
```

```
In [ ]: data = data.drop_duplicates(subset=["context"])
print(len(data))
data.head()

# [
#   {
#     "id": "1",
#     "text": "第1章:データフロント 夜の煌びやかなネオ東京。高層ビルが連なり、ネオンが街を
#             光る。"
#     "source": "file",
#     "source_id": "https://.json",
#     "url": "https://",
#     "created_at": "2021-01-01T12:00:00Z",
#     "author": "著者名"
#   },
#
#   {
#     "id": "1",
#     "text": "第2章:ウルフ・コーポレーションの罠 ミコは、目的地にあるバー「グランマズ・ハウス」
#             で待ち合わせをする。"
#     "source": "file",
#   }
# ]
```

```
#     "source_id": "https://.json",
#     "url": "https://",
#     "created_at": "2021-01-01T12:00:00Z",
#     "author": "著者名"
#   },
#   {
#     "id": "1",
#     "text": "第3章:裏切りと再会 バー「グランマズ・ハウス」で、ミコはデータの受け取り人・リョウ
#     "source": "file",
#     "source_id": "https://.json",
#     "url": "https://",
#     "created_at": "2021-01-01T12:00:00Z",
#     "author": "著者名"
#   }
#
# ]
```

```
In [ ]: documents = [
    {
        'id': r['id'],
        'text': r['context'],
        'metadata': {
            'title': r['title']
        }
    } for r in data.to_dict(orient='records')
]
documents[:3]
```

```
In [ ]: !pip install pinecone-client
```

```
In [ ]: !pip install openai
```

```
In [ ]: import os

BEARER_TOKEN = os.environ.get("BEARER_TOKEN") or "BEARER_TOKEN_HERE"
```

```
In [ ]: headers = {
    "Authorization": f"Bearer {BEARER_TOKEN}"
}
```

```
In [ ]: from tqdm.auto import tqdm
import requests
from requests.adapters import HTTPAdapter, Retry

batch_size = 100
endpoint_url = "http://localhost:8000"
s = requests.Session()

# we setup a retry strategy to retry on 5xx errors
retries = Retry(
    total=5, # number of retries before raising error
    backoff_factor=0.1,
    status_forcelist=[500, 502, 503, 504]
)
s.mount('http://', HTTPAdapter(max_retries=retries))

for i in tqdm(range(0, len(documents), batch_size)):
    i_end = min(len(documents), i+batch_size)
```

```
# make post request that allows up to 5 retries
res = s.post(
    f"{endpoint_url}/upsert",
    headers=headers,
    json={
        "documents": documents[i:i_end]
    }
)
```

```
In [ ]: queries = data['question'].tolist()
# format into the structure needed by the /query endpoint
queries = [{'query': queries[i]} for i in range(len(queries))]
len(queries)
```

```
In [ ]: queries[:3]
```

```
In [ ]: res = requests.post(
    "http://0.0.0.0:8000/query",
    headers=headers,
    json={
        'queries': queries[:3]
    }
)
res
```

```
In [ ]: for query_result in res.json()['results']:
    query = query_result['query']
    answers = []
    scores = []
    for result in query_result['results']:
        answers.append(result['text'])
        scores.append(round(result['score'], 2))
    print("-"*70+"\n"+query+"\n\n"+''.join([f'{s}: {a}' for a, s in zip
```

```
In [ ]: # インデックスへの質問応答
print(chain.run("ミコの幼馴染の名前は?"))
```