第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainとは

最近、LangChainという大規模モデルをエンジンにした全く新しいアプリケーション開発フレームワークが登場し、ほぼChatGPTと同時にリリースされました。プログラマーとして、私たちは大規模モデルの潜在能力とLangChainの便利さを活用して、驚くべき知能アプリケーションを開発することができます。新世代のAI開発フレームワークとして、LangChainはプログラマーの注目を集め、AIアプリケーション開発の新たな熱潮を巻き起こすことでしょう。

これから、LangChainの使い方について一緒に探求していきましょう。それでは、LangChainをどのように理解すればよいのでしょうか？

LangChainは、言語モデルに基づいたアプリケーションを開発するために特化したフレームワークです。LangChainを使用することで、ChatGPT、GPT、Llama などの大規模言語モデルをAPI経由で呼び出すだけでなく、さらに高度な機能を実現することができます。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainの特性

LangChainは、大規模言語モデル（LLM）に基づいたエンドツーエンドの言語モデルアプリケーションを構築するためのフレームワークです。開発者はこのフレームワークを利用して、テキストから画像生成、ドキュメントQA、チャットボットなど、さまざまな複雑なタスクを実現できます。LangChainは、LLMやチャットモデルを活用したアプリケーションの作成を簡素化するために、一連のツール、ライブラリ、インターフェースを提供しています。

真正に有望で革新的なアプリケーションは、単にAPIを通じて言語モデルを呼び出すだけでなく、次の2つの特性を備えることが重要です。

データ感知：言語モデルを他のデータソースと接続し、より豊かで多様なデータを理解し活用できること。

エージェント機能：言語モデルが環境と相互作用し、環境をより深く理解し、効果的に応答できること。

そのため、LangChainフレームワークの設計目標は、このようなAIアプリケーションの実現を目指し、大規模言語モデルの潜在能力を最大限に引き出すことです。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainのインストール

LangChainの基本的なインストールは非常に簡単です。

```
$pip install langchain
```

これはLangChainの最低限のインストール要件です。注意点として、LangChainは様々なモデルやデータリポジトリと統合する必要があります。例えば、最も重要なOpenAIのAPIインターフェースや、オープンソースの大規模モデルライブラリであるHugging Face Hub、さらに様々なベクトルデータベースへのサポートです。しかし、デフォルトでは、これらに必要な依存関係は同時にインストールされません。

つまり、**pip install langchain** を実行した後に、**pip install openai** や、ベクトルデータベースの一つである **pip install chroma** など、追加の依存関係を個別にインストールする必要がある場合があります。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainのインストール

以下の2つの方法で、LangChainをインストールする際にほとんどの依存関係を取り込むことができます。

LangChainをインストールする際に、一般的なオープンソースLLMライブラリを含める方法：

```
$pip install langchain[llms]
```

インストールが完了した後は、LangChainを最新バージョンに更新する必要があります。これにより、新しいツールを使用することができます。

```
$pip install --upgrade langchain
```

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainのインストール

もしソースコードからインストールしたい場合は、リポジトリをクローンして次のコマンドを実行します：

```
$pip install -e
```

Ø　OpenAI API

LangChainを学ぶためには、OpenAIのAPIについて一定の理解が必要です。

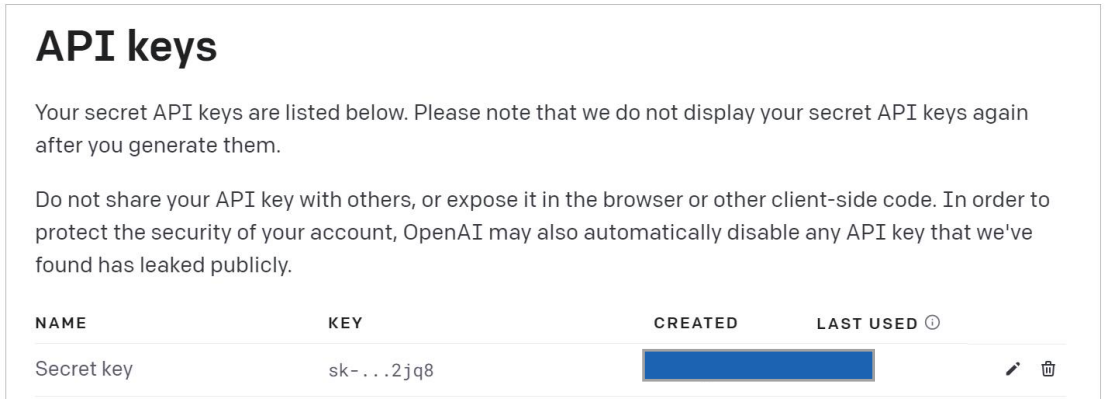
LangChainは、本質的にはさまざまな大規模モデルが提供するAPIのラッパーであり、これらのAPIを使いやすくするために構築されたフレームワーク、モジュール、インターフェースです。

そのため、LangChainの内部ロジックを理解するためには、大規模モデルのAPIの基本的な設計思想を理解する必要があります。現在、最も充実したインターフェースを持ち、最も強力な大規模言語モデルは、OpenAIが提供するGPTファミリーのモデルです。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　OpenAI API

もちろん、OpenAI API を使用するには、まず登録して自分の API キーを取得する必要があります。



OpenAIのアカウントとAPIキーを持っていれば、ダッシュボードでさまざまな情報を見ることができます。例えば、モデルの料金や使用状況などです。そのなかで、さまざまなモデルのアクセス制限情報を示しています。その中で、TPM（tokens-per-minute）とRPM（requests-per-minute）はそれぞれ、1分あたりのトークン数とリクエスト数を意味します。つまり、GPT-4の場合、APIを通じて1分あたり最大で200回の呼び出しと、40,000トークンの転送が可能です。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　OpenAI API

ここで重点的に説明する必要がある2つのモデルは、Chat ModelとText Modelです。これらのモデルは、大規模言語モデルの代表です。もちろん、OpenAIはImage、Audio、その他の種類のモデルも提供していますが、現在のところ、これらはLangChainが主にサポートするモデルではなく、モデルの種類も比較的少ないです。

Chat Model（チャットモデル）は、人間とAIの対話を生成するために使用されるモデルで、代表的なものにはgpt-3.5-turbo（すなわちChatGPT）やGPT-4があります。

Text Model（テキストモデル）は、ChatGPTが登場する前に、GPT-3をAPI経由で呼び出すために使用されていたモデルです。

この2種類のモデルは、機能が似ており、いずれも対話入力（input、またはprompt）を受け取り、回答テキスト（output、またはresponse）を返します。しかし、呼び出し方や要求される入力形式には違いがあります。これについては、後でさらに詳しく説明します。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

以下に、上述の2種類のモデルの呼び出し方法を簡単なコードで説明します。まずは比較的古いTextモデル（GPT-3.5以前のバージョン）を見てみましょう。

Ø　**OpenAI APIでTextモデルの使う方法：**

まず、OpenAIライブラリをインストールします。

```
$pip install openai
```

次に、OpenAI APIキーをインポートします。APIキーをインポートする方法はいくつかありますが、その一つが以下のコードです：

```
import os
os.environ["OPENAI_API_KEY"] = 'あなたのOpen API Key'
```

OpenAIライブラリは、OPENAI_API_KEY という名前の環境変数を確認し、その値をAPIキーとして使用します。また、以下のようにOpenAIライブラリをインポートした後に、api_key の値を指定することもできます。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　**OpenAI APIでTextモデルの使う方法：**

もちろん、キーを直接コードに含める方法は最も避けるべきです。なぜなら、コードをうっかり共有してしまうと、キーが漏洩する可能性があるからです。そのため、より安全な方法は、オペレーティングシステムで環境変数を定義することです。例えば、Linuxシステムのコマンドラインで次のように設定します：

```
$export OPENAI_API_KEY='あなたのOpen API Key'
```

また、環境変数を.envファイルに保存し、python-dotenvライブラリを使ってファイルから読み込む方法もあります。これにより、APIキーがコードに直接記載されるリスクを減らすことができます。

次に、OpenAIライブラリをインポートし、クライアントを作成します。

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

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　OpenAI APIでTextモデルの使う方法：

次に、gpt-3.5-turbo-instruct（すなわちTextモデル）を指定し、completions メソッドを呼び出して結果を取得します。

```
response = client.completions.create(
  model="gpt-3.5-turbo-instruct",
  temperature=0.5,
  max_tokens=100,
  prompt="魯迅と周樹人はどんな関係ですか？"
)
```

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　**OpenAI APIでTextモデルの使う方法：**

OpenAIのテキスト生成モデルを使用する際には、いくつかのパラメータを通じて出力の内容やスタイルを制御することができます。以下に、よく使われるパラメータをまとめました。

1. Temperature（温度）

出力のランダム性を調整します。低い値は安定的、高い値は多様的です。

2. Top-k Sampling

上位k個の単語から次の単語を選びます。多様性を制御します。

3. Top-p Sampling（確率カットオフサンプリング）

累積確率pまでの単語から選択します。動的で柔軟な生成が可能です。

4. Max Tokens（最大トークン数）

生成するテキストの最大長を制限します。

5. Stop Sequences（停止シーケンス）

特定のシーケンスで生成を終了します。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　**OpenAI APIでTextモデルの使う方法：**

次に、モデルが返すテキストを出力します。

```
print(response.choices[0].text.strip())
```

OpenAIのCompletion.createメソッドを呼び出すと、モデルが生成した出力やその他の情報を含むレスポンスオブジェクトが返されます。このレスポンスオブジェクトは、複数のフィールドを持つdict構造になっています。Textモデル（例えばtext-davinci-003）を使用する場合、主なフィールドには以下が含まれます：

1.id: リクエストの一意の識別子。

2.object: オブジェクトの種類（この場合は "text_completion"）。

3.created: リクエストが処理されたUNIXタイムスタンプ。

4.model: 使用されたモデルの名前（例：text-davinci-003）。

5.choices: モデルが生成した結果。複数の選択肢（補完）が含まれ、通常は最も関連性の高いものが最初に表示されます。各選択肢には生成されたテキストが含まれています。

6.usage: リクエストで使用されたトークン数に関する情報。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　**OpenAI APIでTextモデルの使う方法：**

choicesフィールドはリストで、場合によってはモデルに複数の出力を生成するよう要求できます。各選択は辞書であり、以下のフィールドを含みます：

text: モデルが生成したテキスト。

finish_reason: モデルが生成を停止した理由。可能な値には、`stop`（停止マーカーに到達）、`length`（最大長に達した）、または`temperature`（設定された温度パラメータに基づいて停止）があります。

つまり、response.choices[0].text.strip() というコードの意味は以下の通りです：

- レスポンスから最初の選択肢を取得します（nパラメータを指定していない場合、通常は唯一の選択肢しか返されません）。
- その選択肢のテキストを取得します。
- テキストの前後の空白文字を削除します。

これにより、モデルから得られる出力をクリーンにすることができます。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　OpenAI APIでChatモデルの使う方法：

全体の流れとして、ChatモデルとTextモデルの呼び出しは似ていますが、前にchatが追加され、入力（Prompt）と出力（Response）のデータ形式が異なります。

```
response = client.chat.completions.create(
  model="gpt-4",
  messages=[
        {"role": "system", "content": "You are a creative AI."},
        {"role": "user", "content": "魯迅と周樹人はどんな関係です
         か？"},
    ],
  temperature=0.8,
  max_tokens=60
)
```

このコードには、先ほど紹介したtemperatureやmax_tokensなどのパラメータに加えて、Chatモデル専用の2つの概念があります。それは「message」と「role」です。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　**OpenAI APIでChatモデルの使う方法：**

まず「message」について説明します。メッセージはモデルに渡すプロンプトです。ここでの「messages」パラメータはリストで、複数のメッセージを含んでいます。各メッセージには、`role`（`system`、`user`、または`assistant`のいずれか）と`content`（メッセージの内容）が含まれています。システムメッセージは対話の背景を設定します（例：「あなたは素晴らしいAIアシスタントです」）。ユーザーメッセージは具体的なリクエストをします（例：「魯迅と周樹人はどんな関係でか？」）。モデルのタスクは、これらのメッセージに基づいて返信を生成することです。

次に「role」について説明します。OpenAIのChatモデルでは、system、user、assistant の3つのロールがあります。それぞれのロールには異なる意味と役割があります。

system：システムメッセージは、対話の背景やコンテキストを設定するために使われます。これにより、モデルは対話における自分の役割やタスクを理解しやすくなります。

user：ユーザーメッセージは、ユーザーや人間の役割から発信されます。通常、モデルに対して回答やタスクの完了を求める内容が含まれます。

assistant：アシスタントメッセージは、モデルが返信する内容です

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　OpenAI APIでChatモデルの使う方法：

Chatモデルを使用してコンテンツを生成した後、返されるレスポンス（response）には、1つまたは複数の`choices`が含まれます。各`choice`には1つの`message`が含まれており、各`message`も`role`と`content`を含みます。`role`は、メッセージの送信者を示し、`system`、`user`、または`assistant`のいずれかになります。`content`にはメッセージの実際の内容が含まれています。

典型的なresponseオブジェクトは次のようになります：

```
{
 'object': 'chat.completion',
 'model': 'gpt-4',
 'choices': [
   {
    'message': {
      'role': 'assistant',
      'content': '鲁迅と周樹人は同一人物です。'
     },
    'finish_reason': 'stop',
    'index': 0
   }
  ]
}
```

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　Chatモデル vs Textモデル：

ChatモデルとTextモデルにはそれぞれの利点があり、その適用性は具体的なアプリケーションのシナリオに依存します。

Textモデルに比べて、Chatモデルは対話や複数回のインタラクションの処理により適しています。これは、単一の文字列だけでなく、メッセージのリストを入力として受け取ることができるためです。このメッセージリストには、`system`、`user`、`assistant`の歴史的情報が含まれており、インタラクティブな対話を処理する際により多くのコンテキスト情報を提供します。

一般的に見て、Chatモデルはより完成度の高いテキストを提供し、完全な文を出力します。一方、Textモデルは複数の答えなどの断片的な出力をすることがあります。これは、ChatGPTが人間のフィードバックに基づく強化学習を通じて調整されており、そのため出力される回答が実際のチャットシーンにより近いからです。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　Chatモデル vs Textモデル：

Chatモデルの設計による主な利点には以下があります：

対話歴の管理：Chatモデルを使用することで、対話の履歴をより便利に管理でき、必要に応じてこれらの履歴情報をモデルに提供できます。たとえば、過去のユーザー入力やモデルの返信をメッセージリストに含めることで、新しい返信を生成する際にモデルがこれらの歴史的情報を考慮できます。

ロールのシミュレーション：systemロールを使うことで、対話の背景を設定し、モデルに追加の指示情報を提供できます。これにより、出力結果をより良く制御することができます。もちろん、Textモデルでもプロンプト内でAIのロールを設定することができ、入力の一部として利用できます。

OpenAIのAPIについては、これで十分理解できました。結局のところ、私たちは主にLangChainという高度なラッパーフレームワークを通じてOpenAIのモデルにアクセスしています。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainを使用してTextモデルを使う

最後に、LangChainを使用してOpenAIのTextモデルとChatモデルをつかう方法を見てみましょう。

まずはTextモデル、コードは以下の通りです：

```
import os
os.environ["OPENAI_API_KEY"] = 'あなたのOpen API Key'
from langchain.llms import OpenAI
llm = OpenAI(
    model="gpt-3.5-turbo-instruct",
    temperature=0.8,
    max_tokens=60,)
response = llm.predict("魯迅と周樹人はどんな関係でか？")
print(response)
```

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainを使用してTextモデルを使う

これは実際にはOpenAI APIのシンプルなラッパーです。まず、LangChainの`OpenAI`クラスをインポートし、使用するモデルと生成パラメータを指定して`LLM`（大規模言語モデル）オブジェクトを作成します。次に、作成した`LLM`オブジェクトとメッセージリストを使って、`OpenAI`クラスの`__call__`メソッドを呼び出してテキスト生成を行います。生成された結果は`response`変数に格納されます。プログラミング経験がある人にとっては理解しやすく、特に難しい部分はありません。

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainを使用してChatモデルを使う

次に、Chatモデルのコードは以下の通りです：

```
import os
os.environ["OPENAI_API_KEY"] = 'あなたのOpen API Key'
from langchain.chat_models import ChatOpenAI
chat = ChatOpenAI(model="gpt-4",
                  temperature=0.8,
                  max_tokens=60)
from langchain.schema import (
    HumanMessage,
    SystemMessage
)
messages = [
    SystemMessage(content="You are a creative AI."),
    HumanMessage(content="鲁迅と周樹人はどんな関係でか？")
]
response = chat(messages)
print(response)
```

第四章　LangChain: 大規模言語モデルの潜在力を解き放つ利器

Ø　LangChainを使用してChatモデルを使う

このコードも理解しやすいです。主に、LangChainの`ChatOpenAI`クラスをインポートし、Chatモデルのオブジェクトを作成して、使用するモデルと生成パラメータを指定します。その後、LangChainの`schema`モジュールから`SystemMessage`と`HumanMessage`クラスをインポートし、メッセージリストを作成します。このメッセージリストには、1つのシステムメッセージと1つの人間メッセージが含まれています。システムメッセージは通常、コンテキスト設定やAIの振る舞いに関する指示に使われ、人間メッセージはAIに対して応答を求める内容が含まれます。その後、作成した`chat`オブジェクトとメッセージリストを使って、`ChatOpenAI`クラスの`__call__`メソッドを呼び出してテキストを生成します。生成された結果は`response`変数に格納されます。

また、`langchain.llms`の`OpenAI`（Textモデル）や`langchain.chat_models`の`ChatOpenAI`（Chatモデル）で返される`response`変数の構造は、直接OpenAI APIを呼び出すよりも簡潔です。これは、LangChainが大規模言語モデルの出力を解析し、レスポンスの中で最も重要なテキスト部分だけを保持しているためです。