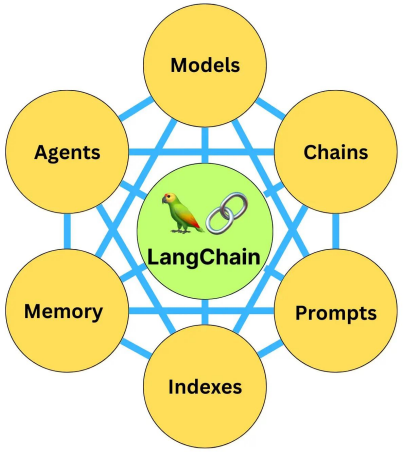
第六章　Model I/O

この章から、LangChainのコアコンポーネントについて順を追って詳細に解析していきます。

ModelはLangChainフレームワークの最下層に位置し、言語モデルに基づいて構築されたアプリケーションの中心要素です。LangChainのアプリケーション開発とは、LangChainをフレームワークとして使用し、APIを通じて大規模なモデルを呼び出して具体的な問題を解決するプロセスです。

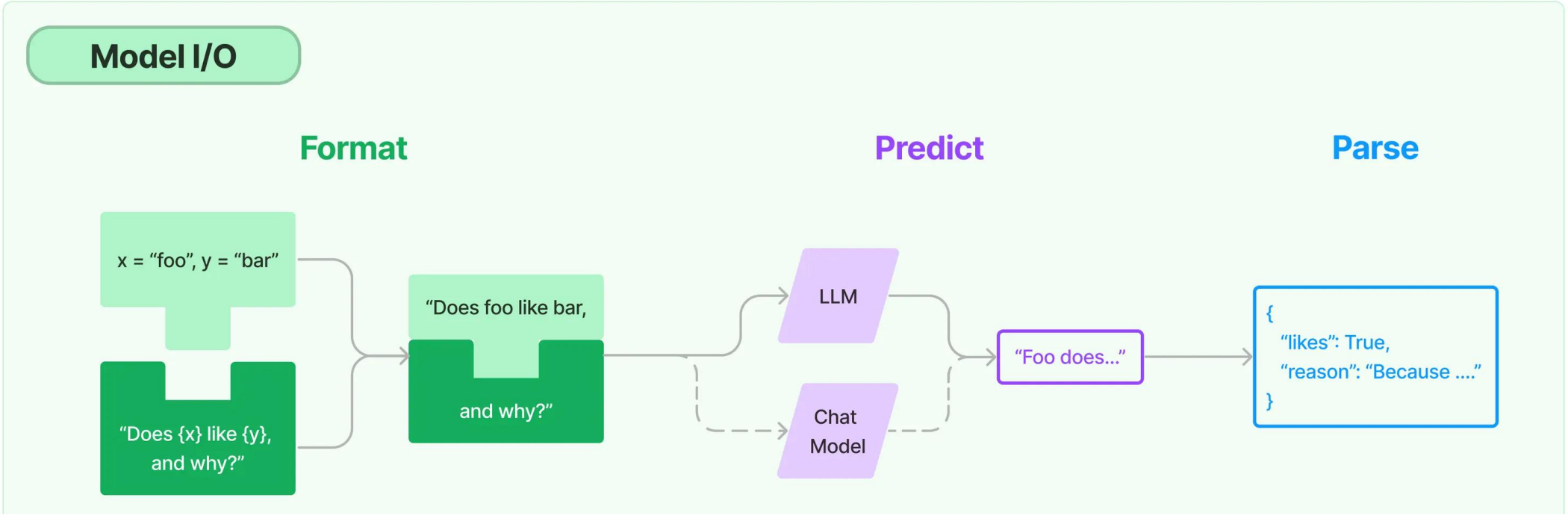
言い換えれば、LangChainフレームワーク全体のロジックはLLMというエンジンによって駆動されています。モデルがなければ、LangChainフレームワークはその存在意義を失うと言えるでしょう。それでは、この章ではモデルについて詳しく説明していきます。



第六章　Model I/O

Ø　Model I/O

モデルの使用プロセスは、次の三つの部分に分解できます。入力提示（図中のFormatに対応）、モデルの呼び出し（図中のPredictに対応）、および出力解析（図中のParseに対応）です。この三つの部分が一体となっており、そのためLangChainではこのプロセス全体を「Model I/O」と呼んでいます。



第六章　Model I/O

Ø　Model I/O

モデル I/O の各段階において、LangChain はテンプレートとツールを提供し、さまざまな言語モデルを迅速に呼び出すためのインターフェースを形成します。

提示テンプレート：モデルを使用する最初のステップは、提示情報をモデルに入力することです。LangChain では、テンプレートを作成して、実際のニーズに応じて異なる入力を動的に選択し、特定のタスクやアプリケーションに合わせて入力を調整することができます。

言語モデル：LangChain では、汎用インターフェースを使用して言語モデルを呼び出すことができます。これにより、どの言語モデルを使用する場合でも、同じ方法で呼び出すことができ、柔軟性と利便性が向上します。

出力解析：LangChain は、モデルの出力から情報を抽出する機能も提供しています。出力解析器を使用することで、モデルの出力から必要な情報を正確に取得でき、冗長または関連性のないデータを処理する必要がありません。さらに、大モデルが返す非構造化テキストを、プログラムが処理できる構造化データに変換することができます。

第六章　Model I/O

Ø　提示テンプレート

言語モデルは無限の宝庫であり、人類の知識と智慧がこの「魔法の箱」に封じ込められているかのようです。しかし、その神秘を解き明かす方法は、まさに知恵のある者とそうでない者の見解に依存しています。そこで、「 Prompt Engineering 」という言葉が特に流行しています。 Prompt Engineeringとは、大規模言語モデルに対するプロンプトの構築を専門に研究する分野です。

とはいえ、大モデルの使用シーンは千差万別であり、すべてのモデルを操ることができる一つまたは二つの魔法のテンプレートが存在するわけではありません。期待通りの回答を得るためには、適切なプロンプトを作成することが非常に重要です。良いプロンプトは、言語モデルの有用性を大幅に向上させることができます。

その具体的な原則は、次の二つに尽きます：

1. **モデルに明確で具体的な指示を与える。**
2. **モデルにじっくり考えさせる**

第六章　Model I/O

Ø　提示テンプレート

言語モデルは無限の宝庫であり、人類の知識と智慧がこの「魔法の箱」に封じ込められているかのようです。しかし、その神秘を解き明かす方法は、まさに知恵のある者とそうでない者の見解に依存しています。そこで、「 Prompt Engineering 」という言葉が特に流行しています。 Prompt Engineeringとは、大規模言語モデルに対するプロンプトの構築を専門に研究する分野です。

とはいえ、大モデルの使用シーンは千差万別であり、すべてのモデルを操ることができる一つまたは二つの魔法のテンプレートが存在するわけではありません。期待通りの回答を得るためには、適切なプロンプトを作成することが非常に重要です。良いプロンプトは、言語モデルの有用性を大幅に向上させることができます。

その具体的な原則は、次の二つに尽きます：

1. **モデルに明確で具体的な指示を与える。**
2. **モデルにじっくり考えさせる**

第六章　Model I/O

Ø　提示テンプレート

例えば、販売している各商品の紹介文を生成したいとします。その場合、社員や顧客が販売中の商品について知りたいときに、このテンプレートを呼び出すことで、モデルがその商品に適した紹介文を生成することができます。

このプロンプトテンプレートの生成方法は次の通りです：

```
from langchain.prompts import PromptTemplate

template = """”
あなたはプロのビジネスコピーライターです。/n
価格が {price} 円の {goods_name} について、魅力的な説明を提供していただけますか？
"""
prompt = PromptTemplate.from_template(template)

print(prompt)
```

第六章　Model I/O

Ø　提示テンプレート

生成されたプロンプトテンプレートの具体的な内容は以下の通りです：

```
input_variables=[‘goods_name', 'price']
output_parser=None partial_variables={}
template='あなたはプロのビジネスコピーライターです。
/n 価格が {price} 円の {goods_name} について、魅力的な説明を提供していただけますか？/n'
template_format='f-string'
validate_template=True
```

ここで言う「テンプレート」とは、特定の商品を説明するためのテキストフォーマットです。これはf-string形式で、{goods_name} と {price} の2つの変数が含まれており、それぞれ商品名と価格を示します。この2つの値はテンプレート内のプレースホルダーであり、実際にテンプレートを使用してプロンプトを生成する際には、具体的な値に置き換えられます。

第六章　Model I/O

Ø　提示テンプレート

コード中の `from_template` はクラスメソッドであり、文字列テンプレートから直接 `PromptTemplate` オブジェクトを作成することができます。この `PromptTemplate` オブジェクトをプリントすると、オブジェクト内の情報には入力変数（この例では `goods_name` と `price`）、出力解析器（この例では指定なし）、テンプレートの形式（この例では 'f-string'）、テンプレートの検証（この例では True に設定）などが含まれます。

したがって、`PromptTemplate` の `from_template` メソッドは、原始的なテンプレート文字列をより豊かで操作しやすい `PromptTemplate` オブジェクトに変換します。このオブジェクトが LangChain におけるプロンプトテンプレートです。LangChain は多くのクラスや関数を提供しており、さまざまなアプリケーションシーンに対応した多くの内蔵テンプレートを設計しています。これにより、プロンプトの構築と使用が容易になります。次回の授業では、プロンプトエンジニアリングの基本原理と LangChain におけるさまざまなプロンプトテンプレートについてさらに詳しく解説します。

次に、先ほど構築したプロンプトテンプレートを使用してプロンプトを生成し、そのプロンプトをモデルに入力します。

第六章　Model I/O

Ø　言語モデル

LangChainでサポートされているモデルには三つの主要なカテゴリがあります。

1. **大規模言語モデル（Text Model）**、これらのモデルはテキスト文字列を入力として受け取り、テキスト文字列を出力として返します。OpenAIの `text-davinci-003`、Facebookの `LLaMA`、ANTHROPICの `Claude` などが典型的なLLMです。

2. **チャットモデル（Chat Model）**、主にOpenAIの `ChatGPT` シリーズが代表的です。これらのモデルは通常言語モデルに基づいていますが、APIはより構造化されています。具体的には、これらのモデルはチャットメッセージのリストを入力として受け取り、チャットメッセージを返します。

3. **テキストEmbeddingモデル（Embedding Model）**、これらのモデルはテキストを入力として受け取り、浮動小数点数のリスト（Embedding）を出力します。例えば、OpenAIの `text-embedding-ada-002` などがこれに該当します。テキストEmbeddingモデルはドキュメントをベクトルデータベースに保存する役割を担っており、プロンプトエンジニアリングとは直接的な関係はありません。

第六章　Model I/O

Ø　言語モデル

次に、言語モデルを呼び出して、モデルに文案の作成を手伝ってもらい、その結果を返してもらいます。

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

from langchain_openai import OpenAI

model = OpenAI(model_name='gpt-3.5-turbo-instruct’)

input = prompt.format(goods_name=[”Play Station 5"], price=‘66000’)

output = model.invoke(input)

print(output)
```

第六章　Model I/O

Ø　言語モデル

以上のコードの中で、input = prompt.format(goods_name=["Play Station 5"], price='66000') というコードの役割は、テンプレートをインスタンス化することです。この時、 {goods_name} が "Play Station 5" に、 {price} が '66000' に置き換えられ、具体的なプロンプトが形成されます:

あなたはプロのビジネスコピーライターです。 価格が 66000円の Play Station 5について、魅力的な説明を提供していただけますか？

第六章　Model I/O

Ø　言語モデル

提示模板を再利用することで、複数の商品プロモーション文案を同時に生成できます。

```
goods = ["PS5", ”Switch", ”XBOX"]
prices = [”66000", "32000", ”55000"]

for goods, price in zip(goods, prices):

    input_prompt = prompt.format(goods_name=goods, price=prices)

    output = model.invoke(input_prompt)

    print(output)
```

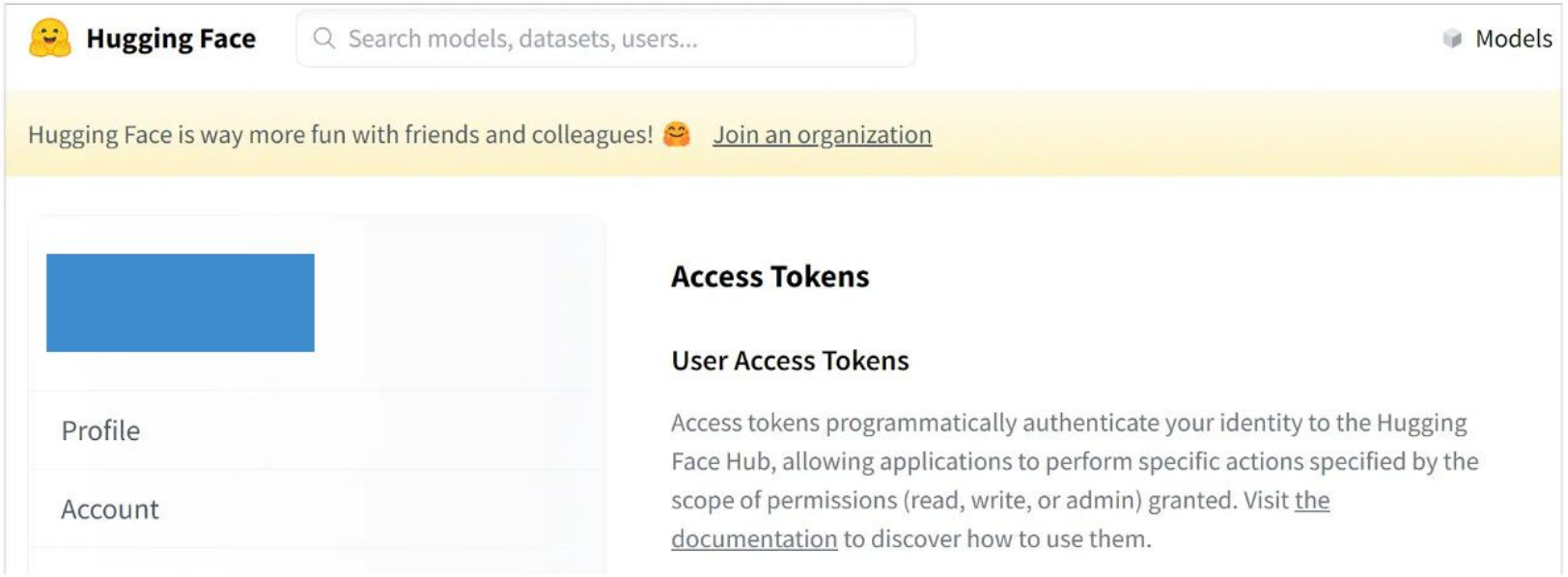
LangChainの強みがここにあることに気づくでしょう。一度テンプレートを定義するだけで、それを使って様々な異なるプロンプトを生成できます。単にf-stringを使ってテキストをフォーマットする方法と比べて、この方法はよりシンプルで、メンテナンスもしやすくなります。また、LangChainのプロンプトテンプレートでは、output_parser、template_format、テンプレートの検証が必要かどうか（validate_template）などの機能も統合されています。

第六章　Model I/O

Ø　言語モデル

さらに、LangChainのプロンプトテンプレートを使用することで、プロンプトに関するコードを一切変更せずに、異なる生成モデルにプログラムを簡単に切り替えることができます。

次に、全く同じプロンプトテンプレートを使用してプロンプトを生成し、HuggingFaceHubのオープンソースモデルに送信して文案を作成します。（注意：HUGGINGFACEHUB_API_TOKENの登録が必要です）



第六章　Model I/O

Ø　言語モデル

ここでお伝えしたい重要な点は、LangChainフレームワークを使用することで、既存のテンプレートやプログラム構造を再利用し、任意のモデルを呼び出すことができるということです。もし機械学習のトレーニングプロセスに精通している場合、LangChainフレームワークはPyTorchやTensorFlowなどのフレームワークと類似していることに気付くかもしれません。これらのフレームワークでは、モデルの選択やトレーニングには高度な柔軟性がありつつ、モデルを呼び出す際のフレームワークは体系化されており、再利用性が高いことが特徴です。

LangChainとプロンプトテンプレートの利点は次の通りです。

コードの可読性: テンプレートにより、複雑なプロンプトも読みやすくなります。

再利用性: テンプレートは複数の場所で使え、コードが簡潔になります。

保守性: テンプレートの修正だけで、コード内のすべての使用箇所を更新できます。

変数処理: テンプレートは変数の挿入を自動で行い、手動での文字列連結が不要です。

パラメータ化: テンプレートは異なるパラメータに応じたプロンプトを生成します。

第六章　Model I/O

Ø　出力解析

LangChainが提供するモデル出力の解析機能により、モデルの出力から構造化された情報をより容易に取得できるようになります。これにより、言語モデルを基盤としたアプリケーション開発の効率が大幅に向上します。

なぜそう言えるのでしょうか？先ほどの例を考えてみてください。モデルに文案を生成させた場合、その結果はテキストの文字列として返されます。これは一見すると必要な情報かもしれませんが、具体的なアプリケーションを開発する際には、単なる文字列だけでは不十分であり、多くの場合、**プログラムが直接処理できる構造化されたデータが必要となります**。

例えば、この文案において、もしモデルに以下の2つのフィールドを返すよう求めるとしましょう。

description：花に関する説明テキスト
reason：上記の文案をどのような理由でそのように書いたのかの説明

第六章　Model I/O

Ø　出力解析

では、曖昧な言語から明確なデータ構造へと自動変換するにはどうすればよいでしょうか？これを実現するのが、LangChainに含まれる「出力解析器」です。

次に、LangChainの出力解析器を使用してプログラムを再構築し、モデルが構造化された応答を生成できるようにします。同時に、その応答を解析し、解析されたデータを直接CSVドキュメントに保存する方法を紹介します。

```
from langchain.output_parsers import StructuredOutputParser, ResponseSchema

response_schemas = [ ResponseSchema(name="description", description="商品の説明"),
ResponseSchema(name="reason", description="なぜこれを書くのですか？") ]

output_parser = StructuredOutputParser.from_response_schemas(response_schemas)

format_instructions = output_parser.get_format_instructions()
```

第六章　Model I/O

Ø　出力解析

上記のコードでは、まず出力の構造を定義しています。ここでは、モデルが生成する回答に「花の説明文（description）」と「この説明文を書く理由（reason）」の2つの部分が含まれるようにしています。そのため、これら2つの出力に対応する **ResponseSchema** オブジェクトを含むリスト **response_schemas** を定義しました。

このリストに基づいて、**StructuredOutputParser.from_response_schemas** メソッドを使用して出力解析器を作成しています。

次に、出力解析器オブジェクトの **get_format_instructions()** メソッドを使用して、出力のフォーマット指示 **format_instructions**を取得します。このフォーマット指示と元の文字列テンプレートを組み合わせて、新しいプロンプトテンプレートを作成します（このテンプレートには出力解析構造の情報が統合されています）。その新しいテンプレートを使ってモデルの入力を生成し、モデルの出力を取得します。この時点で、モデルの出力は可能な限り私たちの指示に従った構造となり、出力解析器が適切に解析できるようになります。

第六章　Model I/O

Ø　出力解析

各商品の価格と組み合わせに対して、**output_parser.parse(output)** を使用して、モデルの出力を事前に定義したデータ形式に解析します。このデータ形式はPythonの辞書で表されており、その辞書には description と reason という2つのフィールドの値が含まれています。

```
output = model.invoke(input)

parsed_output = output_parser.parse(output)

parsed_output['goods'] = goods
parsed_output['price'] = price

df.loc[len(df)] = parsed_output
```

最後に、すべての情報をpandasのDataFrameオブジェクトに統合します。このDataFrameオブジェクトには、goods、price、description、およびreason の4つのフィールドが含まれています。descriptionとreasonは、output_parserがモデルの出力から解析したものであり、goodsとpriceは手動で追加したものです。

第六章　Model I/O

- **出力解析**

DataFrame の内容を出力して確認できるほか、プログラム内で簡単に処理でき、たとえば以下のようにCSVファイルとして保存することも可能です。この時点で、データは曖昧で無構造なテキストではなく、明確なフォーマットを持つ構造化されたデータとなります。ここでの出力解析器の役割は非常に大きいです。

第六章　Model I/O

Ø　まとめ

LangChainフレームワークを使用することで、以下のようなメリットが得られます。

テンプレート管理: 大規模なプロジェクトでは多くの異なるプロンプトテンプレートが存在しますが、LangChainを使用するとこれらのテンプレートを効率的に管理でき、コードの可読性と整然さを保つことができます。

変数抽出とチェック: LangChainはテンプレート内の変数を自動で抽出し、すべての変数が正しく埋められているかを確認します。

モデルの切り替え: 異なるモデルを試したい場合、モデル名を変更するだけで済み、コードの修正は不要です。

出力解析: LangChainのプロンプトテンプレートには出力形式の定義を組み込むことができ、フォーマットされた出力を後続の処理で簡単に扱えるようになります。