Output Parsers in LangChain

1. StrOutputParser

The simplest parser in LangChain, which parses and returns plain text without enforcing any structure. It is useful when we just need free text responses for post-processing.

Example Code:

```
from langchain_huggingface import ChatHuggingFace, HuggingFaceEndpoint
from langchain_core.prompts import PromptTemplate
from langchain_core.output_parsers import StrOutputParser
import os, secrets

os.environ["HUGGINGFACEHUB_API_TOKEN"] = secrets.HuggingFaceHub_ACCESS_TOKEN

llm = HuggingFaceEndpoint(repo_id="openai/gpt-oss-120b", task="text-generation")
model = ChatHuggingFace(llm=llm)

template1 = PromptTemplate(template='Write detail report on the {topic}', input_variables=['topic'])
template2 = PromptTemplate(template='Write down the summary of the given text. /n {text}', input_vari

parser = StrOutputParser()

chain = template1 | model | parser | template2 | model | parser
result = chain.invoke({'topic': 'black hole'})
print(result)
```

Sample Output:

Summary – “Black Holes: A Comprehensive Overview” ... (detailed text response summarizing black holes)

2. JsonOutputParser

This parser generates outputs in JSON format, but does not enforce a schema. We can provide format instructions so that the model outputs structured JSON-like objects.

```
from langchain_core.output_parsers.json import JsonOutputParser
from langchain_huggingface import ChatHuggingFace, HuggingFaceEndpoint
from langchain_core.prompts import PromptTemplate
import os, secrets

parser = JsonOutputParser()

os.environ["HUGGINGFACEHUB_API_TOKEN"] = secrets.HuggingFaceHub_ACCESS_TOKEN

llm = HuggingFaceEndpoint(repo_id="openai/gpt-oss-120b", task="text-generation")
model = ChatHuggingFace(llm=llm)

template = PromptTemplate(
    template = 'Give me the name, age and city of a fictional person \n {format_instruction}',
    input_variables=[],
    partial_variables={'format_instruction' : parser.get_format_instructions()}
)

chain = template | model | parser
result = chain.invoke({})
print(result)
```

Sample Output:

{'name': 'Alex Rivera', 'age': 28, 'city': 'Portland'}

3. StructuredOutputParser

This parser extracts structured JSON data based on a pre-defined schema. It does not enforce strict data validation but ensures that the output contains the requested fields.

```
from langchain.output_parsers import StructuredOutputParser, ResponseSchema
from langchain_huggingface import ChatHuggingFace, HuggingFaceEndpoint
from langchain_core.prompts import PromptTemplate
import os, secrets

os.environ["HUGGINGFACEHUB_API_TOKEN"] = secrets.HuggingFaceHub_ACCESS_TOKEN

llm = HuggingFaceEndpoint(repo_id="openai/gpt-oss-120b", task="text-generation")
model = ChatHuggingFace(llm=llm)

schema = [
    ResponseSchema(name='fact 1', description='Fact 1 of the topic'),
    ResponseSchema(name='fact 2', description='Fact 2 of the topic'),
    ResponseSchema(name='fact 3', description='Fact 3 of the topic')
]

parser = StructuredOutputParser.from_response_schemas(schema)

template = PromptTemplate(
    template = 'Give 3 Facts of the {topic} \n {format_instruction}',
    input_variables=['topic'],
    partial_variables={'format_instruction' : parser.get_format_instructions()}
)

chain = template | model | parser
result = chain.invoke({'topic':'Black Hole'})
print(result)
```

Sample Output:

```
{'fact 1': '...', 'fact 2': '...', 'fact 3': '...'}
```

4. PydanticOutputParser

This parser enforces a strict schema using Pydantic models. It ensures type safety, validation, and seamless integration with downstream systems. This is the most robust way to handle structured outputs.

```
from langchain_huggingface import ChatHuggingFace, HuggingFaceEndpoint
from langchain_core.output_parsers import PydanticOutputParser
from langchain_core.prompts import PromptTemplate
from pydantic import BaseModel, Field
import os, secrets

os.environ["HUGGINGFACEHUB_API_TOKEN"] = secrets.HuggingFaceHub_ACCESS_TOKEN

llm = HuggingFaceEndpoint(repo_id="openai/gpt-oss-120b", task="text-generation")
model = ChatHuggingFace(llm=llm)

class Person(BaseModel):
    name : str = Field(description="Name of the person")
    age : int = Field(gt=18 , description="age of the person")
    city : str = Field(description="name of the city the person belongs to")

parser = PydanticOutputParser(pydantic_object=Person)

template = PromptTemplate(
    template = 'Give the name, age and city of the fictional {place} person \n {format_instruction}'
    input_variables=['place'],
    partial_variables={'format_instruction' : parser.get_format_instructions()}
)

chain = template | model | parser
result = chain.invoke({'place' : 'Pakistan'})
print(result)
```

Sample Output:

```
name='Ayesha Khan' age=27 city='Lahore'
```