Output Parsers in LangChain

Parsing and Structuring LLM Responses Effectively

Overview

If an **LLM supports structured output**, we can use the `with_structured_output()` function directly. However, some LLMs do not natively support structured outputs — in such cases, we rely on **Output Parsers**.

Output Parsers in **LangChain** help convert raw LLM responses into **structured formats** such as JSON, CSV, or Pydantic models. They ensure **consistency, validation, and ease of use** across applications and pipelines.

Why Use Output Parsers?

- To extract structured data when the LLM doesn't support native schemas.
- To validate and enforce consistent field structures.
- To transform raw LLM outputs into usable formats like JSON or Pydantic models.
- To ensure integration with APIs, databases, or downstream tasks.

Popular Output Parsers in LangChain

LangChain provides multiple parser utilities, each with distinct purposes and levels of validation.

Common Parsers

1. **StrOutputParser** — For plain string outputs.
2. **JSONOutputParser** — Returns JSON but without strict structure.
3. **StructuredOutputParser** — Returns structured JSON (no validation).
4. **PydanticOutputParser** — Returns validated structured JSON using Pydantic models.

1. StrOutputParser

This is the simplest parser. It parses the LLM response and returns it as a raw string.

- Useful when building chains where raw text responses are expected.
- Minimal overhead, suitable for logging or intermediate stages.

Example

```
from langchain.output_parsers import StrOutputParser

parser = StrOutputParser()
output = parser.parse("The answer is 42.")
print(output)
# Output: "The answer is 42."
```

2. JSONOutputParser

This parser converts LLM responses into JSON format.

Advantages:

- Simple and lightweight.
- Automatically parses valid JSON text.

Limitations:

- Cannot enforce or validate a specific schema.
- The JSON format is entirely decided by the LLM.

Example

```
from langchain.output_parsers import JSONOutputParser

parser = JSONOutputParser()
output = parser.parse('{"city": "Paris", "country": "France"}')
print(output["city"])
# Output: Paris
```

3. StructuredOutputParser

This parser extracts structured JSON data from LLM responses based on predefined field schemas.

Advantages:

- Allows specifying field names and structure.
- Returns structured JSON.

Limitations:

- No type validation — accepts incorrect types.
- Example: An integer field may accept a string like "7 years".

Example

```
from langchain.output_parsers import StructuredOutputParser, ResponseSchema

schemas = [
    ResponseSchema(name="name", description="Person's name"),
    ResponseSchema(name="age", description="Person's age (int)")
]

parser = StructuredOutputParser.from_response_schemas(schemas)
text = '{"name": "Alice", "age": "7 years"}'
output = parser.parse(text)
print(output)
# Output: {'name': 'Alice', 'age': '7 years'}
```

4. PydanticOutputParser

The most advanced parser — it uses **Pydantic models** to enforce strict schema validation and type safety.

Why Use This:

- Enforces strict schema rules.
- Validates and coerces data types automatically.
- Provides clear validation errors if fields are missing or invalid.

Example

```
from pydantic import BaseModel
from langchain.output_parsers import PydanticOutputParser

class Review(BaseModel):
    summary: str
    sentiment: str
    rating: int

parser = PydanticOutputParser(pydantic_object=Review)
raw_output = '{"summary": "Excellent phone!", "sentiment": "positive",
"rating": "5"}'
structured = parser.parse(raw_output)
print(structured)
# Output: Review(summary='Excellent phone!', sentiment='positive',
rating=5)
```

Feature Comparison

Feature	StrOutputParser	JSONOutputParser	StructuredOutputParser	PydanticOutputParser
Basic parsing	Yes	Yes	Yes	Yes
Schema enforcement	No	No	Partial	Full
Type validation	No	No	No	Yes
Default values	No	No	No	Yes
Error handling	Minimal	Minimal	Basic	Robust
LLM independence	Full	Full	Full	Full

Summary

In summary:

- Use **StrOutputParser** for simple text extraction.
- Use **JSONOutputParser** for plain JSON responses.
- Use **StructuredOutputParser** when you need fixed fields but not validation.
- Use **PydanticOutputParser** for validated, type-safe, and robust structured outputs.

Output Parsers ensure that even when the LLM doesn't support structured output, your application still receives consistent and reliable data.