Structured Output in LLM Applications Tutorial

Part 1 - Prompting

Day 1 of 4

ANSHUMAN JHA

Table of Contents

1. Introduction

Large Language Models (LLMs) like GPT-4 are incredibly powerful tools for generating and processing text.

However, one of the critical challenges in working with LLMs is ensuring that they return structured data in a format that can be easily parsed and used in downstream applications.

This Tutorial, the first in a four-part series, will cover various strategies for returning structured data from LLMs.

In this first part, we'll focus on **Prompting**, a fundamental technique that involves carefully crafting your prompts to guide the LLM's output.

2. High-Level Strategies for Structured Data from LLMs

Before diving into the specifics of prompting, it's essential to understand the broader strategies available for returning structured data from LLMs:

1. **Prompting:** Directly instructing the model to return output in a structured format, such as JSON.
2. **Function Calling:** Leveraging LLM's capabilities to call predefined functions and return structured outputs.
3. **Tool Calling:** Integrating LLMs with external tools or APIs to process and structure data.
4. **JSON Mode:** Using LLM-specific modes that are designed to generate JSON or other structured formats.

3. Setting Up Your Environment

To follow along, you'll need to set up your environment.

We'll be using langchain_openai, a popular library for interacting with OpenAI's language models.

If you haven't installed it yet, you can do so using the following command:

```
!pip install langchain_openai
```

Next, let's import the necessary libraries and initialize the language model.

4. The Importing the LLM

In this example, we'll be using a mini version of GPT-4 called gpt-4o-mini. We'll also use an API key stored in your environment variables.

```
import os
from langchain_openai import ChatOpenAI

# Initialize the language model
llm = ChatOpenAI(model="gpt-4o-mini",
api_key=os.environ.get("OPENAI_API_KEY"))
```

5. Basic Example: Comparing Two Numbers

Before we get into more complex examples, let's start with a simple invocation of the model.

We'll ask the model to compare two numbers.

```
# Invoke the model with a simple question
llm.invoke("Which one is greater, 8.11 or
                                    8.9?")
```

The model will return a response indicating which number is greater.

While this example is straightforward, it demonstrates the basic process of interacting with the LLM.

6. Scenario: Customer Service Conversation

Now, let's move on to a more complex scenario.

Imagine you have a conversation between a customer and a service agent, and you need to extract key information such as the topic of the conversation, the names of the customer and agent, and the sentiment of the call.

Here's the conversation we'll be working with:

```
conversation = '''
Alice: Hi, I bought a laptop from your store recently, but
it's really slow and gets hot quickly. Can you help?

John: Hi Alice, I'm sorry to hear that. I'm here to assist.
Can you tell me the model and when you purchased it?

Alice: It's the XYZ UltraBook 15, and I bought it two weeks
ago.

John: Thanks, Alice. The slowness could be due to
background processes or memory issues. Does it happen with
specific apps?

Alice: Yes, mostly when I’m using video editing software or
have many browser tabs open.

John: That makes sense. Let’s check the Task Manager. Press
Ctrl + Shift + Esc to see which processes are using a lot
of resources.

Alice: I see the video software and something called
“svchost.exe” using a lot of memory.

John: The video software is expected to use a lot, but
svchost.exe might just need a restart to clear it up. How
about the overheating?

Alice: Yes, it gets really hot after a short time.

John: Overheating can be due to blocked vents or high CPU
usage. Try cleaning the vents and placing the laptop on a
hard surface. If it continues, check for updates or bring
it in for a diagnostic.

Alice: Okay, I'll try that. Thanks for your help, John.

John: You’re welcome, Alice! Let me know if you need
anything else. Have a great day!

Alice: Thanks, you too! '''
```

7. Crafting the Prompt

The key to extracting structured data from this conversation lies in the prompt.

We'll craft a prompt that instructs the LLM to extract specific information and return it in JSON format.

```
prompt = f'''You are provided with a
conversation between a customer and an
agent.

Your task is to extract key information
like the topic of the conversation, the
names of the customer and agent, and
the sentiment of the call on a scale of
1 to 5. Always provide results in JSON
format and nothing else.

Conversation: {conversation}'''
```

8. Verifying the Output

Finally, let's verify the type of the output to ensure it is in the expected format.

```
# Verify the type of the output
print(type(out.content))
```

Conclusion

In this first part of the series, we explored how to use **Prompting** to return structured data from LLMs.

By carefully crafting the prompt, we can guide the LLM to produce output in a structured format, such as JSON.

This technique is fundamental and forms the basis for more advanced strategies that we will cover in the following articles.

In the next article, we'll dive into **Function Calling**, where we'll explore how to leverage LLM's capabilities to call predefined functions and return structured outputs. Stay tuned!

Link of collab Notebook

Structured_Output_in_LLM_Applications_Tutorial_Series_Day_1_Promp

August 20, 2024

1 Structured Output in LLM Applications Tutorial Series: Day 1 - Prompting

1.1 Table of Contents

1.2 Table of Contents

---

2 1. Introduction

3 Large Language Models (LLMs) like GPT-4 are incredibly powerful tools for generating and processing text. However, one of the critical challenges in working with LLMs is ensuring that they return structured data in a format that can be easily parsed and used in downstream applications.

#This article, the first in a four-part series, will cover various strategies for returning structured data from LLMs. In this first part, we'll focus on Prompting, a fundamental technique that involves carefully crafting your prompts to guide the LLM's output.

4 2. High-Level Strategies for Structured Data from LLMs

5 Before diving into the specifics of prompting, it's essential to understand the broader strategies available for returning structured data from LLMs:

- **Prompting**: Directly instructing the model to return output in a structured format, such as JSON.
- **Function Calling**: Leveraging LLM's capabilities to call predefined functions and return structured outputs.
- **Tool Calling**: Integrating LLMs with external tools or APIs to process and structure data.
- **JSON Mode**: Using LLM-specific modes that are designed to generate JSON or other structured formats.

#In this article, we will focus on Prompting.

6 3. Setting Up Your Environment

7 To follow along, you'll need to set up your environment.

#We'll be using langchain_openai, a popular library for interacting with OpenAI's language models. #If you haven't installed it yet, you can do so using the following command:

[ ]:
```
!pip install langchain_openai
!pip install langchain_community
```

```
Collecting langchain_openai
  Downloading langchain_openai-0.1.22-py3-none-any.whl.metadata (2.6 kB)
Collecting langchain-core<0.3.0,>=0.2.33 (from langchain_openai)
  Downloading langchain_core-0.2.33-py3-none-any.whl.metadata (6.2 kB)
Collecting openai<2.0.0,>=1.40.0 (from langchain_openai)
  Downloading openai-1.41.1-py3-none-any.whl.metadata (22 kB)
Collecting tiktoken<1,>=0.7 (from langchain_openai)
  Downloading tiktoken-0.7.0-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (6.6 kB)
Requirement already satisfied: PyYAML>=5.3 in /usr/local/lib/python3.10/dist-
packages (from langchain-core<0.3.0,>=0.2.33->langchain_openai) (6.0.2)
Collecting jsonpatch<2.0,>=1.33 (from langchain-
core<0.3.0,>=0.2.33->langchain_openai)
  Downloading jsonpatch-1.33-py2.py3-none-any.whl.metadata (3.0 kB)
Collecting langsmith<0.2.0,>=0.1.75 (from langchain-
core<0.3.0,>=0.2.33->langchain_openai)
  Downloading langsmith-0.1.99-py3-none-any.whl.metadata (13 kB)
Requirement already satisfied: packaging<25,>=23.2 in
/usr/local/lib/python3.10/dist-packages (from langchain-
core<0.3.0,>=0.2.33->langchain_openai) (24.1)
Requirement already satisfied: pydantic<3,>=1 in /usr/local/lib/python3.10/dist-
```

```
packages (from langchain-core<0.3.0,>=0.2.33->langchain_openai) (2.8.2)
Collecting tenacity!=8.4.0,<9.0.0,>=8.1.0 (from langchain-
core<0.3.0,>=0.2.33->langchain_openai)
  Downloading tenacity-8.5.0-py3-none-any.whl.metadata (1.2 kB)
Requirement already satisfied: typing-extensions>=4.7 in
/usr/local/lib/python3.10/dist-packages (from langchain-
core<0.3.0,>=0.2.33->langchain_openai) (4.12.2)
Requirement already satisfied: anyio<5,>=3.5.0 in
/usr/local/lib/python3.10/dist-packages (from
openai<2.0.0,>=1.40.0->langchain_openai) (3.7.1)
Requirement already satisfied: distro<2,>=1.7.0 in /usr/lib/python3/dist-
packages (from openai<2.0.0,>=1.40.0->langchain_openai) (1.7.0)
Collecting httpx<1,>=0.23.0 (from openai<2.0.0,>=1.40.0->langchain_openai)
  Downloading httpx-0.27.0-py3-none-any.whl.metadata (7.2 kB)
Collecting jiter<1,>=0.4.0 (from openai<2.0.0,>=1.40.0->langchain_openai)
  Downloading
jiter-0.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata
(3.6 kB)
Requirement already satisfied: sniffio in /usr/local/lib/python3.10/dist-
packages (from openai<2.0.0,>=1.40.0->langchain_openai) (1.3.1)
Requirement already satisfied: tqdm>4 in /usr/local/lib/python3.10/dist-packages
(from openai<2.0.0,>=1.40.0->langchain_openai) (4.66.5)
Requirement already satisfied: regex>=2022.1.18 in
/usr/local/lib/python3.10/dist-packages (from
tiktoken<1,>=0.7->langchain_openai) (2024.5.15)
Requirement already satisfied: requests>=2.26.0 in
/usr/local/lib/python3.10/dist-packages (from
tiktoken<1,>=0.7->langchain_openai) (2.32.3)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.10/dist-
packages (from anyio<5,>=3.5.0->openai<2.0.0,>=1.40.0->langchain_openai) (3.7)
Requirement already satisfied: exceptiongroup in /usr/local/lib/python3.10/dist-
packages (from anyio<5,>=3.5.0->openai<2.0.0,>=1.40.0->langchain_openai) (1.2.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.10/dist-
packages (from httpx<1,>=0.23.0->openai<2.0.0,>=1.40.0->langchain_openai)
(2024.7.4)
Collecting httpcore==1.* (from
httpx<1,>=0.23.0->openai<2.0.0,>=1.40.0->langchain_openai)
  Downloading httpcore-1.0.5-py3-none-any.whl.metadata (20 kB)
Collecting h11<0.15,>=0.13 (from
httpcore==1.*->httpx<1,>=0.23.0->openai<2.0.0,>=1.40.0->langchain_openai)
  Downloading h11-0.14.0-py3-none-any.whl.metadata (8.2 kB)
Collecting jsonpointer>=1.9 (from jsonpatch<2.0,>=1.33->langchain-
core<0.3.0,>=0.2.33->langchain_openai)
  Downloading jsonpointer-3.0.0-py2.py3-none-any.whl.metadata (2.3 kB)
Collecting orjson<4.0.0,>=3.9.14 (from langsmith<0.2.0,>=0.1.75->langchain-
core<0.3.0,>=0.2.33->langchain_openai)
  Downloading orjson-3.10.7-cp310-cp310-
manylinux_2_17_x86_64.manylinux2014_x86_64.whl.metadata (50 kB)
```

```
                              50.4/50.4 kB
1.4 MB/s eta 0:00:00
Requirement already satisfied: annotated-types>=0.4.0 in
/usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain-
core<0.3.0,>=0.2.33->langchain_openai) (0.7.0)
Requirement already satisfied: pydantic-core==2.20.1 in
/usr/local/lib/python3.10/dist-packages (from pydantic<3,>=1->langchain-
core<0.3.0,>=0.2.33->langchain_openai) (2.20.1)
Requirement already satisfied: charset-normalizer<4,>=2 in
/usr/local/lib/python3.10/dist-packages (from
requests>=2.26.0->tiktoken<1,>=0.7->langchain_openai) (3.3.2)
Requirement already satisfied: urllib3<3,>=1.21.1 in
/usr/local/lib/python3.10/dist-packages (from
requests>=2.26.0->tiktoken<1,>=0.7->langchain_openai) (2.0.7)
Downloading langchain_openai-0.1.22-py3-none-any.whl (51 kB)
                              52.0/52.0 kB
2.3 MB/s eta 0:00:00
Downloading langchain_core-0.2.33-py3-none-any.whl (391 kB)
                              391.5/391.5 kB
14.1 MB/s eta 0:00:00
Downloading openai-1.41.1-py3-none-any.whl (362 kB)
                              362.5/362.5 kB
16.8 MB/s eta 0:00:00
Downloading
tiktoken-0.7.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (1.1
MB)
                              1.1/1.1 MB
27.6 MB/s eta 0:00:00
Downloading httpx-0.27.0-py3-none-any.whl (75 kB)
                              75.6/75.6 kB
3.7 MB/s eta 0:00:00
Downloading httpcore-1.0.5-py3-none-any.whl (77 kB)
                              77.9/77.9 kB
4.4 MB/s eta 0:00:00
Downloading
jiter-0.5.0-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (318 kB)
                              318.9/318.9 kB
17.0 MB/s eta 0:00:00
Downloading jsonpatch-1.33-py2.py3-none-any.whl (12 kB)
Downloading langsmith-0.1.99-py3-none-any.whl (140 kB)
                              140.4/140.4 kB
5.8 MB/s eta 0:00:00
Downloading tenacity-8.5.0-py3-none-any.whl (28 kB)
Downloading jsonpointer-3.0.0-py2.py3-none-any.whl (7.6 kB)
Downloading
orjson-3.10.7-cp310-cp310-manylinux_2_17_x86_64.manylinux2014_x86_64.whl (141
kB)
                              141.9/141.9 kB
```

```
8.9 MB/s eta 0:00:00
Downloading h11-0.14.0-py3-none-any.whl (58 kB)
                        58.3/58.3 kB
3.4 MB/s eta 0:00:00
Installing collected packages: tenacity, orjson, jsonpointer, jiter, h11,
tiktoken, jsonpatch, httpcore, langsmith, httpx, openai, langchain-core,
langchain_openai
  Attempting uninstall: tenacity
    Found existing installation: tenacity 9.0.0
    Uninstalling tenacity-9.0.0:
      Successfully uninstalled tenacity-9.0.0
Successfully installed h11-0.14.0 httpcore-1.0.5 httpx-0.27.0 jiter-0.5.0
jsonpatch-1.33 jsonpointer-3.0.0 langchain-core-0.2.33 langchain_openai-0.1.22
langsmith-0.1.99 openai-1.41.1 orjson-3.10.7 tenacity-8.5.0 tiktoken-0.7.0
```

#Next, let's import the necessary libraries and initialize the language model.

8 4. Importing the LLM

#In this example, we'll be using a mini version of GPT-4 called gpt-4o-mini. We'll also use an API key stored in your environment variables.

[ ]:
```python
import os
import openai
from langchain_openai import ChatOpenAI
from functools import lru_cache

# Set your OpenAI API key here
os.environ["OPENAI_API_KEY"] = "your_OpenAI_API_key"

llm = ChatOpenAI(model="gpt-4o-mini", api_key=os.environ.get("OPENAI_API_KEY"))
```

9 5. Basic Example: Comparing Two Numbers

#Before we get into more complex examples, let's start with a simple invocation of the model. We'll ask the model to compare two numbers.

[ ]:
```python
# Invoke the model with a simple question
llm.invoke("Which one is greater, 8.11 or 8.9?")
```

[ ]:
```
AIMessage(content='8.9 is greater than 8.11.', additional_kwargs={'refusal':
None}, response_metadata={'token_usage': {'completion_tokens': 11,
'prompt_tokens': 22, 'total_tokens': 33}, 'model_name':
'gpt-4o-mini-2024-07-18', 'system_fingerprint': 'fp_48196bc67a',
'finish_reason': 'stop', 'logprobs': None},
id='run-247bf29f-cba2-4301-86f9-6f962b19e0fe-0', usage_metadata={'input_tokens':
22, 'output_tokens': 11, 'total_tokens': 33})
```

#The model will return a response indicating which number is greater. #While this example is straightforward, it demonstrates the basic process of interacting with the LLM.

10 6. Scenario: Customer Service Conversation

#Now, let's move on to a more complex scenario. Imagine you have a conversation between a customer and a service agent, and you need to extract key information such as the topic of the conversation, the names of the customer and agent, and the sentiment of the call. Here's the conversation we'll be working with:

[ ]:
```
conversation = '''
Alice: Hi, I bought a laptop from your store recently, but it's really slow and␣
 ↪gets hot quickly. Can you help?
John: Hi Alice, I'm sorry to hear that. I'm here to assist. Can you tell me the␣
 ↪model and when you purchased it?
Alice: It's the XYZ UltraBook 15, and I bought it two weeks ago.
John: Thanks, Alice. The slowness could be due to background processes or␣
 ↪memory issues. Does it happen with specific apps?
Alice: Yes, mostly when I'm using video editing software or have many browser␣
 ↪tabs open.
John: That makes sense. Let's check the Task Manager. Press Ctrl + Shift + Esc␣
 ↪to see which processes are using a lot of resources.
Alice: I see the video software and something called "svchost.exe" using a lot␣
 ↪of memory.
John: The video software is expected to use a lot, but svchost.exe might just␣
 ↪need a restart to clear it up. How about the overheating?
Alice: Yes, it gets really hot after a short time.
John: Overheating can be due to blocked vents or high CPU usage. Try cleaning␣
 ↪the vents and placing the laptop on a hard surface. If it continues, check␣
 ↪for updates or bring it in for a diagnostic.
Alice: Okay, I'll try that. Thanks for your help, John.
John: You're welcome, Alice! Let me know if you need anything else. Have a␣
 ↪great day!
Alice: Thanks, you too!
'''
```

11 7 Crafting the Prompt

#The key to extracting structured data from this conversation lies in the prompt. We'll craft a prompt that instructs the LLM to extract specific information and return it in JSON format.

[ ]:
```
# Craft the prompt for extracting structured data
prompt = f'''You are provided with a conversation between a customer and an␣
 ↪agent.
```

```
Your task is to extract key information like the topic of the conversation, the␣
 ↪names of the customer and agent, and the sentiment of the call on a scale of␣
 ↪1 to 5.
Always provide results in JSON format and nothing else.
Conversation: {conversation}'''
```

12 8. Invoking the Model with the Prompt

#Now, let's invoke the model with the crafted prompt. We will also use a callback function to monitor the API usage.

[ ]:
```
from langchain_community.callbacks import get_openai_callback

# Invoke the model with the crafted prompt
with get_openai_callback() as cb:
    out = llm.invoke(prompt)
print(out.content)
print(cb)
```

```
```json
{
  "topic": "Laptop performance issues",
  "customer_name": "Alice",
  "agent_name": "John",
  "sentiment": 4
}
```
Tokens Used: 413
        Prompt Tokens: 374
        Completion Tokens: 39
Successful Requests: 1
Total Cost (USD): $7.95e-05
```

###Explanation:

1. Prompt Construction: The prompt is carefully constructed to instruct the model to return only the requested information in JSON format. This is crucial for ensuring that the output is structured and usable.
2. Callback Function: The get_openai_callback function is used to track the API usage and gather information about the request, which can be helpful for optimization and cost management.
3. Model Invocation: The invoke method sends the prompt to the model and retrieves the structured output.

13 9. Verifying the Output

14 Finally, let's verify the type of the output to ensure it is in the expected format.

[ ]:
```
# Verify the type of the output
print(type(out.content))
```

```
<class 'str'>
```

15 6. Conclusion

#In this first part of the series, we explored how to use Prompting to return structured data from LLMs. By carefully crafting the prompt, we can guide the LLM to produce output in a structured format, such as JSON. This technique is fundamental and forms the basis for more advanced strategies that we will cover in the following articles.

#In the next article, we'll dive into Function Calling, where we'll explore how to leverage LLM's capabilities to call predefined functions and return structured outputs. Stay tuned!