Structured Output in LangChain

1. Introduction to Structured Output

The core concept of **structured output** in LangChain, a crucial advancement in how Large Language Models (LLMs) interact with other systems. Traditionally, LLMs like ChatGPT generate "unstructured output," which is typically plain text. This makes programmatic interaction with databases, APIs, or other software systems difficult because "text में कोई स्ट्रक्चर नहीं होता" (text has no structure).

Structured output enables LLMs to return responses in a "well defined data format" such as JSON, making the model's output "easier to pass and work with programmatically." The primary benefit is the ability to "integrate these LLMs with other systems very easily," moving beyond simple human-LLM interaction to enable LLM-machine communication. This is a "very important topic" for future concepts like Agents.

2. Unstructured vs. Structured Output Illustrated

Unstructured Output Example: When asked "What is the capital of India?", an LLM typically responds with "New Delhi is the capital of India." This is pure text and lacks a predefined format.

Structured Output Example: For a prompt like "Can you create a one-day travel itinerary for Paris?", an LLM, when configured for structured output, could return a JSON object like this:

```
[
{
"time": "Morning",
"activity": "Visit Eiffel Tower"
},
{
"time": "Afternoon",
"activity": "Visit a museum"
},
{
"time": "Evening",
"activity": "Have dinner"
}
]
```

This JSON format provides a clear structure with specific keys (time, activity) and values, making it machine-readable and parsable.

3. Key Use Cases for Structured Output

Structured output offers significant advantages across various applications. Three primary use cases are highlighted:

- **Data Extraction:**
- **Description:** Extracting specific, structured information from unstructured text (e.g., resumes, documents) for storage in databases.

- **Example:** In a job portal like Naukri.com, a candidate uploads a resume. An LLM can extract key information (name, last company, 10th/12th/college marks) in JSON format, which can then be directly inserted into a database. "आप स्ट्रक्चर्ड आउटपुट को यूज कर सकते हो" (You can use structured output).
- **API Building:**
- **Description:** Creating APIs where the output of an LLM needs to be consumed by other services in a structured way.
- **Example:** For an e-commerce platform like Amazon, product reviews (which are long and unstructured) can be processed by an LLM to extract "topic," "pros," "cons," and "overall sentiment" in a structured format. This structured data can then be exposed via a Flask or FastAPI API for other applications to consume.
- **Building Agents:**
- **Description:** Agents are "chatbots on steroids" that can perform actions beyond just chatting, often by interacting with tools. These tools typically require structured input.
- **Example:** A math-based agent might have a "calculator" tool. If a user asks "Find the square root of two," the text "Find the square root of two" cannot be directly sent to the calculator. Structured output can extract the *operation* (square root) and the *number* (two) as distinct, structured pieces of information, which the calculator tool can then process. "यह जो भी टूल्स आप यूज करते हो एजेंट्स के साथ इन सबको स्ट्रक्चर्ड आउटपुट की जरूरत होती है" (All these tools that you use with agents require structured output).

4. Implementing Structured Output in LangChain: with_structured_output Function

LangChain provides the with_structured_output function to facilitate structured output from LLMs. This function handles the underlying mechanics of generating an appropriate system prompt to guide the LLM.

- **Core Mechanism:** When with_structured_output is called with a defined schema, LangChain internally generates a "system prompt" (e.g., "You are an AI assistant that extracts structured insights from text...") that instructs the LLM to return information in the specified JSON format. The original user prompt is then attached, and the combined prompt is sent to the LLM.
- **Model Compatibility:**
- **Models with Built-in Support:** Some LLMs, like OpenAI's GPT models, are "by default structured output generate kar sakte hain" (can generate structured output by default). For these, with_structured_output simplifies the process significantly.
- **Models Without Built-in Support:** For LLMs that do not natively support structured output (e.g., certain Hugging Face models like TinyLlama), LangChain's Output Parsers are used. This topic will be covered in a subsequent video.

5. Defining Data Formats (Schemas) with with_structured_output

The with_structured_output function accepts schemas defined in three main ways:

- **a) Typed Dictionaries (TypedDict):**
- **Concept:** A Python construct (from the typing module) that allows defining a dictionary's expected keys and their value types. It serves as a "type hint" for developers, ensuring a "specific structure" for dictionaries.
- **Benefit:** Provides type hinting in code editors, improving code readability and preventing type-related errors *during development*.
- **Limitation:** "कोई गारंटी नहीं है कि अगर आपने यहां पर बोल दिया कि समरी शुड बी अ स्ट्रिंग तो पलट के स्ट्रिंग ही आएगा" (There's no guarantee that if you say here that the summary should be a string, it will return a string). It offers no runtime validation; if an incorrect type is assigned, the code will still run without error.

- **Use Case:** When "You only need type hints" and your entire project uses Python, without needing to share the schema across different programming languages.
- **b) Pydantic Models:**
- **Concept:** A robust data validation and parsing library for Python. Pydantic models inherit from BaseModel and allow defining fields with specific types.
- **Key Advantage:** Provides **runtime data validation**. If the input data does not conform to the defined schema (e.g., an integer is provided where a string is expected), Pydantic will raise an error, stopping the code execution. "अगर डेटा हमारी बात नहीं मान रहा तो हम एरर थ्रो कर पा रहे हैं कोड को रोक पा रहे हैं" (If the data doesn't follow our instructions, we can throw an error and stop the code).
- **Additional Features:** Supports default values, optional fields, implicit type conversion (type coercing), built-in validators (e.g., for email addresses), and custom constraints using Field (e.g., range validation for numbers, regex patterns).
- **Use Case:** When "आपको डेटा वैलिडेशन चाहिए होता है" (you need data validation) or default values, or automatic type conversion. This is considered the "go-to format" for Python-based LLM applications.
- **c) JSON Schema:**
- **Concept:** A standardized, language-agnostic format for describing the structure of JSON data.
- **Key Advantage: Cross-language compatibility.** Since JSON is universally understood, a JSON Schema can be used across multiple programming languages (e.g., Python backend, JavaScript frontend). "JSON is a universal data format, it can be understood by any language."
- **Structure:** Includes title, description, type (e.g., object, array), properties (for attributes), and required fields.
- **Use Case:** When your project involves "multiple languages" and you need to share the schema definition across them.

6. with_structured_output Methods: json_mode vs. function_calling

The with_structured_output function has a method parameter that dictates how the structured output is generated:

- **json_mode:**
- **Purpose:** To receive structured output directly in JSON format.
- **Recommendation:** Use with models like Claude or Gemini that support JSON structured output.
- **function_calling:**
- **Purpose:** To receive structured output specifically to call a function. This is particularly relevant for "Agents" that need to invoke "tools." The LLM generates arguments for the function call in a structured format.
- **Recommendation:** Use with OpenAI models, as this is often their default and recommended method for structured output.