bi connect

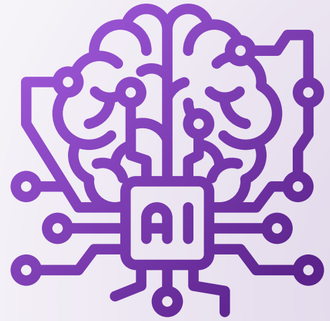
Structured Data Extraction using Instructor in LLM

Presented By

Balamathankumar
Farook

TALENTSHIP | octonomy

Table of Contents



Understanding Structured Data Extraction

- Data extraction is evolving with the emergence of Large Language Models (LLMs)
- Structured data extraction is the process of converting unstructured or semi-structured information into a pre-defined, organized format (Eg: JSON, XML)

How It Differs from Unstructured Data:

- Unstructured: Free-form, inconsistent, difficult to parse.
- Structured: Consistent, machine-readable, easier to integrate.

Key Concepts

Schema, data fields, type safety, and validation

Example of Structured vs. Unstructured Output

❌ Unstructured Output (Hard to Parse)

- The product "Premium Plan" costs $29.99 and has a duration of 12 months. It includes unlimited mileage.

✅ Structured Output (Easy to Parse)

- { "product_name": "Premium Plan", "price": 29.99, "duration_months": 12, "mileage": "unlimited" }

The Need for Structured Data Extraction

01

Enhanced Consistency & Reduced Ambiguity

Clear, organized output formats improve readability and ease of integration

02

Improved Usability

Structured data (e.g., JSON, XML) is easier for downstream applications to parse and process

03

Organized Data for Analysis

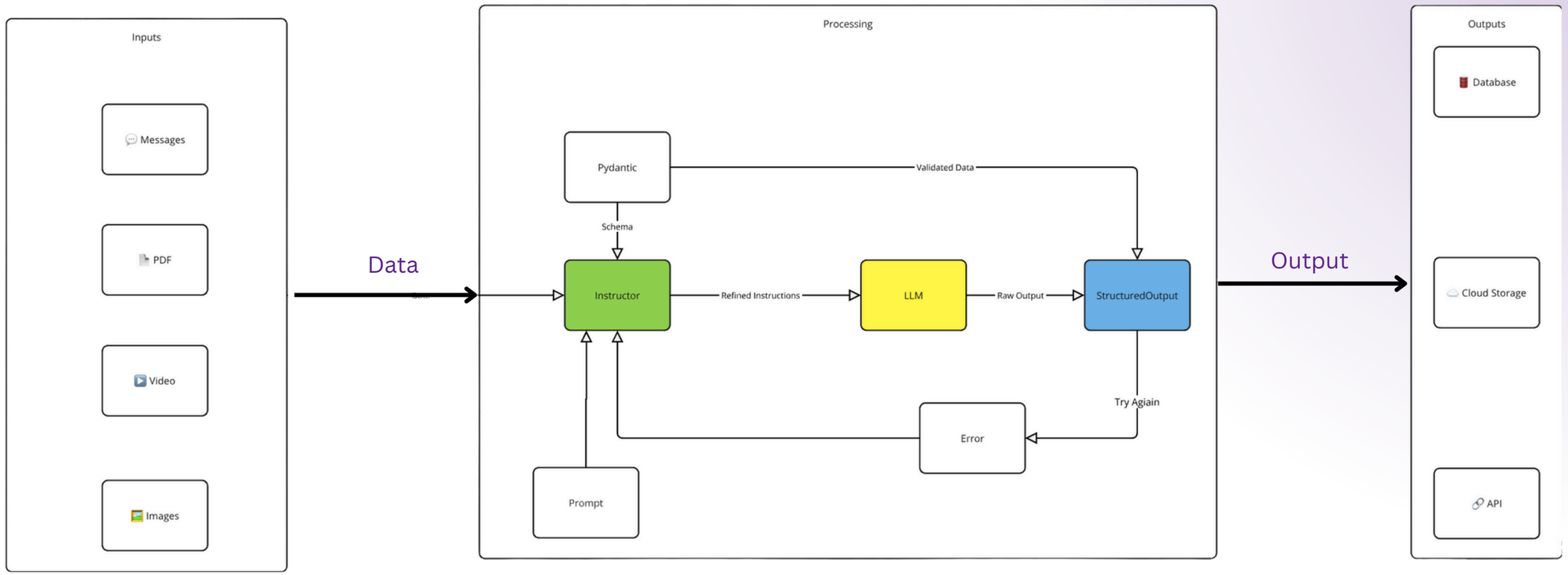
Facilitates generating reports, summaries, or data-driven insights

04

Reliability in Applications

Critical for domains like legal documentation, financial reporting, or real-time systems

Structured Data Extraction



Approaches to extract information using LLMs

Tool/Function Calling Mode

Some LLMs support a *tool or function calling* mode. These LLMs can structure output according to a given schema.
Generally, this approach is the easiest to work with and is expected to yield good results

JSON Mode

Some LLMs are can be forced to output valid JSON. This is similar to tool/function Calling approach, except that the schema is provided as part of the prompt.
Generally, our intuition is that this performs worse than a tool/function calling approach, but don't trust us and verify for your own use case!

Prompting Based

LLMs that can follow instructions well can be instructed to generate text in a desired format. The generated text can be parsed downstream using existing Output Parsers or using custom parsers into a structured format like JSON. This approach can be used with LLMs that do not support JSON mode or tool/function calling modes.
This approach is more broadly applicable, though may yield worse results than models that have been fine-tuned for extraction or function calling

Approaches to extract information using LLMs

Function Calling Mode

```python
import openai
import json

def extract_info(text):
    messages = [ {"role": "user", "content": f"Extract name, age, and occupation from: {text}"}]
    functions = [
        {
            "name": "extract_info",
            "description": "Extracts name, age, and occupation.",
            "parameters": {
                "type": "object",
                "properties": {
                    "name": {"type": "string"},
                    "age": {"type": "number"},
                    "occupation": {"type": "string"}
                },
                "required": ["name", "age", "occupation"]
            }
        }
    ]

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo-0613",
        messages=messages,
        functions=functions,
        function_call={"name": "extract_info"}
    )

    args = response.choices[0].message.get("function_call", {}).get("arguments", "{}")
    return json.loads(args)

text = "John Doe is a 29-year-old software engineer."
data = extract_info(text)
print(data)
```

JSON Schema in Prompt

```python
import openai

prompt = """
Extract the following details and output as JSON.
Schema:
{
  "name": string,
  "age": number,
  "occupation": string
}

Text: John Doe is a 29-year-old software engineer.
"""

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}]
)

print(response.choices[0].message.content)
```

Explicit Formatting Instructions

```python
# This example instructs the LLM to output data as key-value pairs.
import openai

prompt = (
    "Extract the following details as key-value pairs:\n"
    "Name:\nAge:\nOccupation:\n\n"
    "Text: John Doe is a 29-year-old software engineer."
)

response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=[{"role": "user", "content": prompt}]
)

print(response.choices[0].message.content)
```

Introducing Instructor

- Instructor is a specialized Python library designed to enhance interactions with Large Language Models (LLMs) by providing structured data extraction capabilities

- It acts as a bridge between raw LLM outputs and strongly-typed Python objects, ensuring data consistency and reliability

- The package was developed to address the challenge of extracting specific, structured information from LLM responses in a type-safe manner

- Instructor supports a wide range of AI model providers, each with their own capabilities and features, all integration.

Key Benefits of Instructor

Simple API & Full Prompt Control	Complete ownership over prompts allows fine-tuned customization
Reasking & Validation	Automatically re-asks the model if outputs fail validation, using Pydantic for robust error handling
Streaming Support	Enables real-time processing by streaming partial results
Powered by Type Hints	Leverages Pydantic to enforce schema validation and reduce boilerplate code
Broad LLM Provider Support	Compatible with OpenAI, Anthropic, Google, Vertex AI, Mistral/Mixtral, Ollama, and others

Realtime Usecases

Real-Time Data Integration

APIs and databases require consistent, structured data

Use Cases

- Financial Reporting: Immediate data extraction from financial documents
- Legal Documentation: Reliable extraction of key data points
- Dashboards & Analytics: Real-time updates for decision-making

Impact of Unstructured Data

- Increased error rates and manual post-processing
- Delays in data-driven decision-making

Limitations of Structured Outputs

Potential Rigidity

May limit the flexibility of responses

Performance Trade-Offs

Sometimes, ensuring strict structure may impact the creative aspects or reasoning capabilities of the model

Complexity in Prompt Engineering

Designing effective prompts for structured outputs can be challenging

Implementation Overhead

Requires additional setup and maintenance, especially in dynamic environments

Conclusion

QnA

THANK YOU