Structured Outputs from Large Language Models

Seth J. Chandler, with help from AI

August 16, 2025

1 Introduction

When working with Large Language Models (LLMs) in legal practice, you will often need to move beyond generating human-readable text. To truly harness the power of AI, you must command it to produce information that other software can process. This is the domain of **structured outputs**.

Consider the difference between a witness's deposition testimony and a standardized Civil Cover Sheet form required for filing a new case. The testimony is an unstructured narrative—fluid, detailed, and rich with context, but difficult for a computer to parse. The cover sheet, in contrast, is pure structure. It contains discrete, labeled fields: "Parties," "Cause of Action," "Jurisdiction." Both convey critical information, but the cover sheet is designed for immediate, unambiguous, and automated data entry into a court's case management system.

Structured outputs are the LLM's version of that cover sheet. They are responses formatted to follow specific, predictable rules, making them **machine-readable** while remaining human-interpretable. By mastering the ability to request and use structured outputs, you transform an LLM from a sophisticated writing partner into the central engine of a highly automated and efficient legal workflow.

2 Why Structured Outputs are a Game-Changer in Legal Practice

The core advantage of using structured data is that it bridges the gap between the nuanced, linguistic world of an LLM and the rigid, logical world of traditional computer programs. Software that runs your case management system, billing software, or compliance checklists operates with mathematical precision and executes tasks thousands of times faster than an LLM can "think." By converting an LLM's insights into structured data, you empower these powerful, specialized tools.

2.1 Precision and Speed Advantages

When an LLM extracts key dates, party names, or financial figures from a 100-page contract and presents them in a structured format, you can then use other software to instantly:

- **Automate Data Entry:** Automatically populate client information, case details, and deadlines into your firm's case management system, eliminating manual entry and reducing clerical errors.
- **Generate Reports:** Instantly create compliance reports, discovery logs, or summaries of contract terms across dozens of documents.
- **Build Searchable Databases:** Convert insights from thousands of pages of discovery documents into a database you can query with surgical precision (e.g., "Show me all emails sent by 'John Smith' to 'Jane Doe' in March 2024").
- **Perform Quantitative Analysis:** Analyze billing records to identify trends, or review a portfolio of loan agreements to calculate total financial exposure under specific conditions.

2.2 Workflow Integration and Scalability

Legal professionals rely on an ecosystem of interconnected software. Structured outputs act as the universal translator, allowing your AI assistant to communicate seamlessly with your existing legal technology stack. This enables a level of automation and scale previously unimaginable.

Imagine you need to review 500 commercial lease agreements for a specific indemnification clause. A junior associate might take weeks to complete this task. An LLM, prompted correctly, could extract the relevant clauses and format them as structured data in minutes. A simple script could then analyze that data to categorize each lease, flag those needing immediate review, and even draft initial client notification emails. This is the power of combining AI's language capabilities with the logic of software, all made possible by structured outputs.

3 How to Generate Structured Output: Just Ask

Before we get mired in the details of structured output, it is important to appreciate just how easy the process has become. The most fundamental and often most effective way to get structured output from a large language model is to simply ask for it directly. This technique is the foundation of prompt engineering for structured data. By appending a clear instruction at the end of your prompt, you can guide the model to format its response in a way that is immediately usable by other software.

This method works for nearly any standard format. You can be direct and specific in your request:

- "Summarize the key findings of this judicial opinion and **please write your output as Markdown**."
- "Draft a motion to compel based on the attached discovery requests. **Please write your output as raw LaTeX code that I can cut and paste into Overleaf**."
- "Extract the parties, filing date, court, and damages sought from this complaint. **Please structure your answer as a single JSON object**."

This straightforward approach should always be your starting point. For many tasks, it's all you need to bridge the gap between a conversational response and machine-readable data.

3.1 Using the User Interface: The Canvas or Artifact Window

Modern LLM interfaces, like Gemini and Claude, often include a feature that makes working with structured outputs even easier. When you ask for a specific format like code, JSON, or Markdown, the AI may place its response in a separate, dedicated window next to the chat. This feature is sometimes called a **Canvas** (Gemini) or an **Artifact** (Claude).

This is incredibly useful because it:

1. **Isolates the Output:** It cleanly separates the structured data from the conversational text, so you don't have to manually edit out phrases like "Certainly, here is the JSON you requested...".
2. **Prevents Errors:** It ensures that you can copy and paste the entire block of code or data with one click, reducing the risk of missing a bracket or a comma.
3. **Facilitates Editing:** The dedicated window often acts as a mini-editor, allowing you to make small changes to the output before copying it to its final destination.

When you need structured data, explicitly asking for it and leveraging these UI features creates a clean and efficient workflow. If only all of life was this simple.

4 The Leading Structured Output Formats

Several formats have become standard for structuring data. Each has its strengths and is suited to different tasks.

4.1 JSON (JavaScript Object Notation)

JSON is the de facto standard for data exchange in modern web applications and APIs. It is lightweight, easy for humans to read, and trivial for machines to parse.

- **What it does:** JSON organizes data into `key-value` pairs. A "key" is a label (like "Case Name"), and a "value" is the data (like "Smith v. Jones").

- **History:** The basic concepts were in use in the late 1990s, but JSON was formally specified and popularized by Douglas Crockford in the early 2000s.
- **Example:**

```
{
  "case_name": "Smith v. Jones",
  "case_number": "2024-CV-12345",
  "parties": { "plaintiff": "John Smith", "defendant": "Jane
   Jones" }
}

```

- **Legal Use Cases:** The default choice for sending data to or from case management systems, legal APIs, or any modern software. Ideal for contract term extraction, client intake processing, and legal research compilation.

4.2 CSV (Comma-Separated Values)

CSV is the simplest format for representing tabular data. It is the native language of spreadsheet programs like Microsoft Excel and Google Sheets.

- **What it does:** CSV creates a table where each line is a row and commas separate the values in each column. The first line is typically the header row.
- **History:** The concept has been in use since the earliest days of computing, with widespread adoption in the 1970s. It remains popular due to its simplicity.
- **Example:**

```
Case Name,Case Number,Filing Date,Status
Smith v. Jones,2024-CV-12345,2024-03-15,Active
Brown v. Green,2024-CV-12346,2024-03-20,Settled

```

- **Legal Use Cases:** Managing case dockets, client contact lists, time and billing records, and preparing data for statistical analysis.

4.3 YAML (YAML Ain't Markup Language)

YAML prioritizes human readability, using indentation to represent structure. This makes it excellent for data that humans need to write or edit frequently.

- **What it does:** YAML organizes data hierarchically using indentation, avoiding the brackets and braces of JSON.
- **History:** First released in 2001, it was designed to be more readable than other data formats like XML.

- **Example:**

```yaml
case_details:
  title: "Smith v. Jones"
  number: "2024-CV-12345"
parties:
  plaintiff:
    name: "John Smith"
  defendant:
    name: "Jane Jones"

```

- **Legal Use Cases:** Creating complex case timelines, outlining multi-party litigation structures, and writing configuration files for legal automation workflows.

4.4 XML (eXtensible Markup Language)

XML is a verbose and strictly defined format that uses tags to define data elements. While less common in new development, it remains a cornerstone of many enterprise and government systems.

- **What it does:** XML wraps data in opening and closing tags, like `<title>Smith v. Jones</title>`.
- **History:** Developed by the World Wide Web Consortium (W3C) in 1996, it was designed to be a more flexible and robust successor to HTML for data exchange.
- **Example:**

```xml
<case_info>
  <title>Smith v. Jones</title>
  <number>2024-CV-12345</number>
</case_info>

```

- **Legal Use Cases:** Integrating with court e-filing systems (like those using the LegalXML standard), regulatory reporting, and processing e-discovery data that uses XML "load files."

4.5 Markdown

Markdown is a lightweight markup language for creating formatted text using a plain-text editor. Its goal is readability, both in its raw and rendered forms.

- **What it does:** Markdown uses simple symbols (`#` for a heading, `*` for a bullet point) to add structure to documents.
- **History:** Created in 2004 by John Gruber and Aaron Swartz as a simpler way to write content for the web.

- **Example:**

```
# Case Summary: Smith v. Jones
## Key Facts
- Plaintiff alleges breach of contract.
- Defendant failed to deliver goods on time.

```

- **Legal Use Cases:** Drafting legal briefs, memos, client updates, and internal research notes. It is excellent for quickly creating well-formatted documents without complex word processing software.

4.6 LaTeX

LaTeX is a powerful document preparation system for creating high-quality, professionally typeset documents. It is not just a markup language but a full programming environment for typography.

- **What it does:** LaTeX uses commands (e.g., `\section{Introduction}`) to define not just the structure, but the precise layout, typography, and formatting of a document.
- **History:** LaTeX was created in 1985 by Leslie Lamport, built upon the TeX typesetting system developed by Donald Knuth in 1978. It is ancient by software standards but remains unparalleled in its domain.
- **Example:**

```
\documentclass{article}
\begin{document}

\section{Introduction}
This case, \textit{Smith v. Jones}, involves a breach of
  contract...

\end{document}

```

- **Legal Use Cases:** Generating perfectly formatted court pleadings that adhere to strict rules (margins, line numbering, fonts), authoring academic legal articles, and creating complex documents with cross-references, tables of contents, and indices.

4.7 Overleaf: Making LaTeX Accessible

For decades, using LaTeX required a significant technical hurdle: installing and maintaining a complex local software environment (a TeX distribution). However, the arrival of **Overleaf** has fundamentally changed the landscape and made LaTeX accessible to a much wider audience. Overleaf is a cloud-based, collaborative LaTeX editor that runs entirely in your web browser. This is a game-changer because it eliminates the need for any local installation.

It provides a user-friendly interface with the raw LaTeX code on one side and a real-time preview of the compiled PDF on the other, allowing you to see changes as you make them. Furthermore, its Google Docs-style collaboration features allow multiple authors to work on a single document simultaneously, making it an indispensable tool for academic and professional teams drafting complex documents like legal briefs, scholarly articles, or formal reports.

5 Choosing the Right Format: A Practical Guide

The format you choose depends entirely on the **end use** of the data.

Table 1: A Practical Guide for Choosing a Structured Format

If your goal is...	**Then choose...**	**Why?**
Data for another program or API to use	**JSON**	It's the universal language of modern software. Lightweight, easy for any programming language to handle. This is your default choice for automation.
Organizing data in a simple table or spreadsheet	**CSV**	It's the simplest possible format for tabular data and opens directly in Excel or Google Sheets for analysis or manual review.
Creating a human-readable document (memo, report, email)	**Markdown**	It's incredibly easy to write and read. It focuses on clean, semantic structure without getting bogged down in formatting details.
Producing a print-quality, formally typeset document	**LaTeX**	It gives you absolute control over the final appearance, which is essential for court filings or academic publishing. It's powerful but complex.
Data that a human needs to easily read and edit by hand	**YAML**	Its indentation-based structure is often more intuitive than JSON's braces and quotes, making it ideal for configuration files or complex outlines.
Integrating with a legacy system or government agency	**XML**	Use it when you are forced to. If a court's e-filing system or an old document management platform requires XML, that is what you must provide.

In short: Start with **JSON** for automation and **Markdown** for documents. Use the others when a specific need arises.

6 Advanced Tools for Reliable Extraction

Prompting an LLM is powerful, but for professional, scalable workflows, you need to guarantee that the output is always correct. Advanced tools help enforce these guarantees.

6.1 Pydantic: The Data Schema Enforcer

Pydantic is a Python library that acts as a strict quality-control inspector for your data. You define the "shape" of your desired output in a Python class, including the exact field names and data types (e.g., `filing_date` must be a date, `damages_sought` must be a number).

```python
from pydantic import BaseModel, Field
from datetime import date
from typing import List

class LegalCase(BaseModel):
    case_name: str
    case_number: str
    filing_date: date
    parties: List[str]
    damages_sought: float | None = Field(default=None)
```

Listing 1: Define the exact structure you expect from the LLM

When you get a JSON output from an LLM, you feed it to your Pydantic model.

- **If the JSON is perfect:** Pydantic parses it into a clean, usable Python object.
- **If the JSON is flawed** (e.g., the date is in the wrong format, a required field is missing, or a number is provided as text), Pydantic immediately raises an error, telling you exactly what is wrong.

This prevents bad data from corrupting your downstream systems and makes your automation pipeline robust.

7 Advanced Tools for Reliable Extraction

Prompting an LLM is powerful, but for professional, scalable workflows, you need to guarantee that the output is always correct. Advanced tools build a robust framework around the LLM to enforce rules and handle errors, turning a creative model into a predictable data processing tool.

8 Advanced Tools for Reliable Extraction

For professional, scalable workflows, you need to guarantee that an LLM's output is always correct. Advanced tools build a robust framework around the LLM to enforce rules and handle errors, turning a creative model into a predictable data processing tool.

8.1 Google's LangExtract: The Production-Grade Extraction Engine

LangExtract is a Python library from Google designed for one specific task: pulling structured data from unstructured text reliably and at scale. It acts as an end-to-end engine, handling the complex tasks of prompt optimization, interacting with the LLM, and validating the output. This is in contrast to a validation library like **Pydantic**, which is a component used to check data rather than an entire extraction system.

LangExtract is particularly powerful because it formalizes the "few-shot learning" approach. You provide a clear prompt and high-quality examples, and the library uses them to reliably guide the LLM.

8.1.1 Example: Setting Up a Citation Hallucination Checker

To build a tool that checks legal citations, the first step is to reliably extract every proposition and its corresponding citation from a document. Here is how you would configure LangExtract for this task using its prompt-based approach.

```python
import langextract as lx
import textwrap

# 1. Define the guiding prompt for the LLM
prompt = textwrap.dedent("""\
    Extract legal propositions and their corresponding citations.
    For each finding, create a ’citation_record’ with attributes for the
    proposition, the full citation text, and the parsed volume, reporter,
    and page numbers.
""")

# 2. Provide a high-quality example using the lx.data classes
examples = [
    lx.data.ExampleData(
        text="The court established that mere negligence is not sufficient to prove liability, see Smith v. Jones, 123 U.S. 456.",
        extractions=[
            lx.data.Extraction(
                extraction_class="citation_record",
                extraction_text="The court established that mere negligence is not sufficient to prove liability, see Smith v. Jones, 123 U.S. 456.",
                attributes={
                    "proposition": "The court established that mere negligence is not sufficient to prove liability.",
                    "citation_text": "Smith v. Jones, 123 U.S. 456",
                    "volume": 123, "reporter": "U.S.", "page": 456
                }
            )
        ]
    )
]

# 3. Initialize the extractor
# llm = MyLLM() # Your chosen language model
# extractor = lx.Extractor(llm, prompt=prompt, examples=examples)
```

Listing 2: Configuring LangExtract with a prompt and examples.

8.1.2 Using the Extracted Data: A Pseudocode Workflow

Once LangExtract runs, it returns a list of structured 'Extraction' objects. The crucial next step is to use the clean data from the 'attributes' of these objects to perform the validation logic. The workflow is shown in the following pseudocode.

```
// 1. Run the extractor on the document text
structured_citations = extractor.extract(document_text)

// 2. Initialize a list to hold the final results
final_report = []

// 3. Loop through each piece of structured data
FOR EACH citation_object IN structured_citations:

    // Get the clean data from the object's attributes
    attributes = citation_object.attributes

    // Call an external legal API (e.g., CourtListener)
    canonical_case = CALL_LEGAL_API(
        volume: attributes.volume,
        reporter: attributes.reporter,
        page: attributes.page
    )

    // Check the result from the API
    IF canonical_case EXISTS THEN
```

```
        // If the case is real, use an LLM to analyze it
        support_status = CALL_LLM_ANALYZER(
            proposition: attributes.proposition,
            case_text: canonical_case.text
        )

        // Store the positive result
        ADD {
            citation: attributes.citation_text,
            found: TRUE,
            status: support_status
        } TO final_report

    ELSE
        // If the case was not found, mark it as such
        ADD {
            citation: attributes.citation_text,
            found: FALSE,
            status: "NOT_FOUND"
        } TO final_report
    END IF

END FOR

// 4. Display the compiled results
DISPLAY_JSON(final_report)
```

Listing 3: Pseudocode for the citation checker's logic.

This workflow highlights the power of structured outputs. The initial, messy task of parsing a legal brief is handled by LangExtract. This produces predictable, structured data, which then enables a simple, logical program to perform a series of complex validation and analysis steps.

9 Visualizing and Converting Structured Data

Raw structured data can be hard to read. Fortunately, excellent tools exist to help.

9.1 Visualization Tools

- **Text Editors:** Modern text editors like **Visual Studio Code**, Sublime Text, and Atom are essential. They provide **syntax highlighting**, which color-codes your data to make it instantly readable. They also have tools to automatically format ("pretty-print") and validate your files, catching simple errors like a missing comma.

- **Spreadsheet Programs: Microsoft Excel** and **Google Sheets** are the best visualizers for CSV data. They can also import JSON and XML data, though the process can be more complex.
- **Online Viewers and Browser Extensions:** Numerous websites and browser plugins (like JSONView for Chrome/Firefox) can take raw JSON or XML and display it as a clean, collapsible, and searchable tree structure. This is perfect for quickly inspecting an LLM's output.

9.2 Conversion Tools: The Power of Pandoc

Often, you'll need to convert from one format to another. For documents, the undisputed champion is **Pandoc**. It is a command-line "Swiss Army knife" that can convert between dozens of formats.

With Pandoc, you can instantly:

- Turn a **Markdown** file into a perfectly formatted **Microsoft Word** document (`.docx`).
- Convert a **Markdown** file into a **PDF** (by using a LaTeX engine in the background).
- Transform a simple text file into an **HTML** web page.

For data formats (JSON, CSV, YAML), conversion is typically handled by small scripts using programming libraries or by using specialized online tools. For example, you can easily find a tool to convert a CSV of cases into a list of JSON objects for use in an application.