Unstructured to Structured Pipeline

DSC 360 F25 Lab 5 mini
Professor Thomas E. Allen

Omkar and Chenny

Overview

In this lab, we are trying to build a structured extraction model, using large language models, and pydantic module from Python. We are going to work with unstructured data (text) as input and use LLM's to find meaningful parts out of it and differentiate between them semantically where meaning is important. We are storing the structured output in a data frame for further use.

Problem

A lot of times we are working with big documents. Will we have to find a similar kind of information for different periods are different sections. For example, if I must find past years financial numbers for company, I will have to go to the 10 K's (annual reports) an input everything in excel manually. This has a very good chance of making errors. It also takes tremendous amount of time.

Methods

This also is a very specific task and large language models are very good at small and specific tasks. We are trying to leverage the tools function of Ollama, where we can enforce a kind of schema for our output from LLM. Large language models are very likely to hallucinate and to avoid that we must give exactly the information we are looking for a small chunk to the model. In general, we can achieve this by using our RAG pipeline.

For this experiment, we have our input files (.txt). Each line represents a class at Central College. It has all the information about the class like the schedule, the professor, the topic, the subject and etc.. We are just giving each line to our LLM model and enforcing the schema that be built using pydantic. This schema has all the requirements of each variable and for even more constraining validations we have regular expressions and data conversion.

We used two different models:

granite3.2:2b, a smaller version, which will have good speed because of less parameters
llama3.1:8b, a bigger and popular model for more accuracy

We use two different prompts
Prompt 1:

Extract all structured course information from the given text line.

Always return output strictly as a single JSON object following the exact schema below — no extra text or commentary.

Input text:
{line}

Schema and rules:
```
{{
 "program": string,              # Exactly 3 uppercase letters (e.g., CSC, MAT, ECO). If not found, return null.
 "number": string,               # 3 digits optionally followed by 'L' (e.g., 210, 210L). If not found, return null.
  "section": string or null,       # Single lowercase letter (e.g., 'a') or null if missing.
  "title": string or null,         # Full course title.
  "credits": float or null,        # e.g., 3.0, 4.0. Must be numeric.
  "days": string or null,          # Pattern like '-M-W-F-' or '--T-R--'. Return null if it shows '-------'.
  "times": string or null,         # e.g., '9:00-9:50AM'. Return null if 'TBA'.
  "faculty": string or null,       # Instructor’s full name.
  "room": string or null,          # Format 'BUILDING ROOM' (e.g., 'OLIN 208'). Return null if 'TBA'.
  "tags": string or null           # Optional classification codes separated by commas, e.g., 'E1,A'. Null if none.
}}
```

Important rules:
- Always return JSON with double quotes around property names and string values.
- Never include explanations or comments in output.
- If something is missing or unclear, set the value to null rather than omitting the key.
- Preserve capitalization and punctuation in text fields like 'title' and 'faculty'.
- Do not guess field values; derive them only from the input text.

Example Input:
"CSC 210L a Intro to Data Science 4.0 -M-W-F- 10:00-10:50AM Smith, John OLIN 208 E1,A"

Example Output:
```
{{
  "program": "CSC",
  "number": "210L",
  "section": "a",
  "title": "Intro to Data Science",
  "credits": 4.0,
  "days": "-M-W-F-",
  "times": "10:00-10:50AM",
  "faculty": "Smith, John",
  "room": "OLIN 208",
  "tags": "E1,A"
}}
```

Prompt 2:

Extract structured course information from this one-line input:

\"\"\"{line}\"\"\"

Return the output ONLY as a JSON object with the following fields:

```
{{
  "program": string or null,       # Exactly 3 uppercase letters, e.g., "CSC", "MAT"
  "number": string or null,        # Three digits optionally followed by 'L', e.g. "210", "210L"
  "section": string or null,       # Single lowercase letter, e.g., "a", or null if missing
```

```
 "title": string or null,         # Full course title as a string
 "credits": number or null,       # Numeric, e.g., 3, 4 (integer or float)
 "days": string or null,          # Days pattern like "-M-W-F-", "--T-R--", or null if "-------"
 "times": string or null,         # Time range like "9:40-11:10AM" or null if "TBA"
 "faculty": string or null,       # Instructor's full name or initials, e.g. "A. Hor"
 "room": string or null,          # Always "BUILDING ROOM" format, e.g. "OLIN 208", null if "TBA"
 "tags": string or null           # Optional tags, comma separated, e.g. "E1,A"
}}

Important rules:
- The JSON output must have all keys exactly as above, with null for missing or unknown fields.
- The "program" must NEVER contain building names or room numbers.
- The "room" must not include tags or faculty names.
- The "tags" field should contain only classification codes, never building or room info.
- Follow exact casing and format shown in examples.
- Do not combine or merge multiple fields.
- If uncertain about a field, set it explicitly to null.

Example input line:
"MUS 120 Materials & Structure of Music I 3 a Z. Klobnak 12:40-2:10PM --T-R-- GRNT 403 E1, A"

Expected JSON output:
{{
 "program": "MUS",
 "number": "120",
 "section": "a",
 "title": "Materials & Structure of Music I",
 "credits": 3,
 "days": "--T-R--",
 "times": "12:40-2:10PM",
 "faculty": "Z. Klobnak",
 "room": "GRNT 403",
 "tags": "E1, A"
}}
Return ONLY the JSON object, nothing else.
```

Analysis

We went through a lot of scratch prompts to build onto the prompt one and two. Giving examples of what all the possibilities are there for this variable work well it's solved some of the validation problems for example, when we said only three letters for program, it stopped confusing program for building (which often had 4 letters). Also giving an example of how it should look helped a lot of errors where something was just completely missing. Giving Constraints of what not to do sometimes helped.

Over defining the rules was not really helping sometimes it was fitting the rule itself as the variable. Since prop two had a rule section, this made the overall accuracy better, but there was not even one which was completely correct, so the role level accuracy was zero. We have to still figure out how to better this.

Below are results from the experiments with mixture of prompt and model.

Metric	Prompt1 &Model 1	Prompt1 &Model 2	Prompt2 &Model 1	Prompt2 &Model 2
program	1.00	1.00	1.00	1
number	0.94	0.32	0.00	0.98
section	0.08	0.08	0.10	0.78
title	0.82	0.80	0.66	0.70
credits	0.84	0.80	0.72	0.74
days	0.18	0.22	0.42	0.40
times	0.12	0.22	0.48	0.36
room	0.80	0.84	0.84	0.92
faculty	0.88	0.98	0.98	0.98
tags	0.80	0.80	0.80	0.82
Row-level EM	0.0	0.0	0.00	0.00

Conclusion

This experiment was very useful, as mentioned this is a very applicable project in our everyday tasks. If we had more time, we would do this for financial modeling, where all the past financial report from a company are input in the model and we get a CSV file or data frame, which will have everything in a structured format to analyze the data.

References

https://docs.pydantic.dev/latest/

https://docs.ollama.com/capabilities/structured-outputs#python-2

https://www.w3schools.com/python/python_regex.asp

https://youtu.be/XIdQ6gO3Anc

https://youtu.be/dJsTkj9xCKA