Generative structured data extraction using LLMs

Mara Wilhelmi, Sherjeel Shabih, Martino Rıos Garcıa, Santiago Miret,Christoph Koc

4/11/24

Table of contents

1 Generative structured data extraction using LLMs

Structured data is at the heart of machine learning. LLMs offer a convenient way to generate structured data based on unstructured inputs. This book gives hands-on examples of the different steps in the extraction workflow using LLMs.

2 Constrained generation to guarantee syntactic correctness

Note

If we want to generate output that is structured in a specific way, we can use various techniques to

- make the extraction more efficient (but automatically adding the "obvious" tokens)
- make the generation guaranteed to be syntactically correct
- make the generation sometimes more semantically correct, too

To enable constrained decoding, we will use one of the most popular packages for this task `instructor`. It is built on `pydantic` and can leverage function calling and JSON-mode of the OpenAI API as well as other constrained sampling approaches.

```
from pydantic import BaseModel, Field
from typing import List, Optional, Literal
import erdantic as erd
import instructor
from IPython.display import SVG
from openai import OpenAI
from dotenv import load_dotenv
load_dotenv('../.env', override=True)
```

```
True
```

2.1 Defining a data schema

For most constrained generation tasks, we need to define a data schema in a programmatic way. The most common way to do so is to use `pydantic` data classes. Here is an example of a simple data schema for a recipe:

```
from pydantic import BaseModel

class Recipe(BaseModel):
    title: str
    ingredients: List[str]
    instructions: List[str]
```

This schema can also be extended to include descriptions of different fields or to only allow certain values for specific fields. For example, we could add a field for the number of servings and only allow positive integers.

```
from pydantic import BaseModel, Field
from typing import Literal, List

class Recipe(BaseModel):
    title: str
    ingredients: List[str]
    instructions: List[str]
    servings: int = Field(..., gt=0, description="The number of servings for this recipe")
    rating: Literal["easy", "medium", "hard"] = Field("easy", description="The difficulty
```

If we want to extract copolymerization reactions a data schema could look like the following.

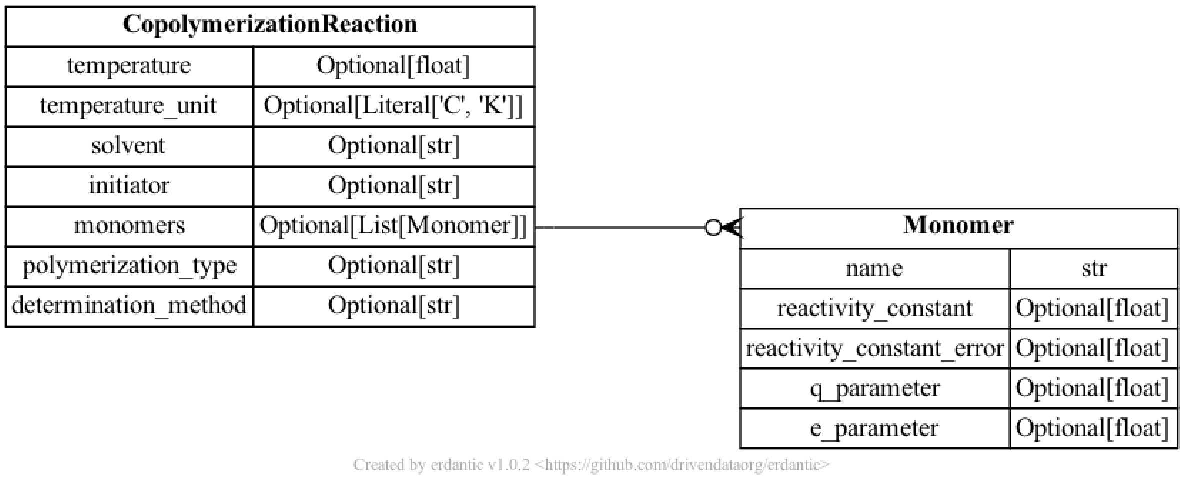
We can now use `instructor` to "patch" the OpenAI API client to ensure that our output fulfils the schema.

```
client = instructor.patch(OpenAI(), mode=instructor.Mode.MD_JSON)


class Monomer(BaseModel):
    name: str = Field(..., title="Name", description="Name of the monomer.")
    reactivity_constant: Optional[float] = Field(
        None,
        title="Reactivity constant",
        description="Reactivity constant of the monomer. r1 for monomer 1 and r2 for monom
        ge=0,
    )
    reactivity_constant_error: Optional[float] = Field(
        None,
        title="Reactivity constant error",
        description="Error in the reactivity constant. Often indicated with +/-. Must be g
        ge=0,
    )
    q_parameter: Optional[float] = Field(
        None,
        title="Q parameter",
        description="Q parameter of the monomer. Q1 for monomer 1 and Q2 for monomer 2. Mu
        ge=0,
    )
    e_parameter: Optional[float] = Field(
        None,
        title="e parameter",
        description="e parameter of the monomer. e1 for monomer 1 and e2 for monomer 2.",
    )


class CopolymerizationReaction(BaseModel):
    temperature: Optional[float] = Field(
        ...,
        title="Temperature",
        description="Temperature at which the reaction is carried out",
    )
    temperature_unit: Optional[Literal["C", "K"]] = Field(
        ..., title="Temperature unit", description="Unit of temperature"
    )
    solvent: Optional[str] = Field(
        None,
        title="Solvent",
        description="Solvent used in the reaction. If bulk polymerization was performed, t
    )
    initiator: Optional[str] = Field(
        None, title="Initiator", description="Initiator used in the reaction"
    )
    monomers: Optional[List[Monomer]] = Field(
        ...,
        title="Monomers",
        description="Monomers used in the reaction. Ensure that the reactivity ratios are
        min_items=2,
        max_items=2,
    )
    polymerization_type: Optional[str] = Field(
        ...,
        title="Polymerization type",
        description="Type of polymerization (e.g., bulk, solution, suspension, emulsion)",
    )
    determination_method: Optional[str] = Field(
        ...,
        title="Determination method",
        description="Method used to determine the reactivity ratios (e.g. Kelen Tudor, Fin
    )
```

```
diagram = erd.create(CopolymerizationReaction)
diagram.draw('diagram.png')
```


CopolymerizationReaction
temperature | Optional[float]
temperature_unit | Optional[Literal['C', 'K']]
solvent | Optional[str]
initiator | Optional[str]
monomers | Optional[List[Monomer]]
polymerization_type | Optional[str]
determination_method | Optional[str]
Monomer
name | str
reactivity_constant | Optional[float]
reactivity_constant_error | Optional[float]
q_parameter | Optional[float]
e_parameter | Optional[float]
Created by erdantic v1.0.2 <https://github.com/drivendataorg/erdantic>

In this case, we will use PDF files in the form as images as input for the model. To perform this conversion, we import some utilities.

```
from pdf2image import convert_from_path
from utils import process_image, get_prompt_vision_model
```

The code below only converts each page of the PDF into an image and then generates dictionary objects in a format that can be used by the OpenAI API.

```
filepath = 'paper01.pdf'
pdf_images = convert_from_path(filepath)

images_base64 = [process_image(image, 2048, 'images', filepath, j)[0] for j, image in enum
images = get_prompt_vision_model(images_base64=images_base64)
```

Armed with the images, we can now use the OpenAI API to extract the text from the images. For this, we just call the API with our prompts and the images.

```
completion = client.chat.completions.create(
    model="gpt-4-turbo",
    response_model=List[CopolymerizationReaction],
    max_retries=2,
    messages=[
        {
            "role": "system",
            "content": """You are a scientific assistant, extracting accurate information
Do not use data that was reproduced from other sources.
If you confuse the reactivity ratios with other numbers, you will be penalized.
Monomer names might be quite similar, if you confuse them, you will be penalized.
NEVER combine data from different reactions, otherwise you will be penalized.
If you are unsure, return no data. Quality is more important than quantity.
""",
        },
        {
            "role": "user",
            "content": """Extract the data from the paper into the provided data schema. W
The relationship between monomers and parameters is typically indicated by subscripts that
Never return data that you are not absolutely sure about! You will be penalized for incorr
        },
        {"role": "user", "content": [*images]},
    ],
    temperature=0,
)
```

```
completion
```

```
[CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='carbon tetrachlor
 CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='chloroform', initi
 CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='acetone', initiato
 CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='1,4-dioxane', ini
 CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='acetonitrile', ini
```

3 OCR with Nougat

To run machine learning models on papers, we need to convert them into plain text. This might require parsing PDFs, and sometimes optical character recognition (OCR).

This is a difficult problem as not only the content of the paper needs to be extracted, but also the layout can be important for the interpretation of the paper. In addition, scientific papers often contain mathematical formulas, which are not easy to parse.

To address those issues, researchers from Meta have trained the Nougat system, which takes PDF as input and produces Markdown as output.

3.1 Cleaning the data

For certain applications it might be necessary to clean the data before using it for other downstream tasks.