Generative structured data extraction using LLMs

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

2024-04-11

Table of contents

1 Generative structured data extraction using LLMs

1.1 About this book

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.

Part I

Data extraction workflow

2 1 | Obtaining data for data extraction

At the start of each data extraction process one has to first collect a dataset containing all data sources relevant for the extraction topic. Various Python packages are available to make this process easier and more efficient.

- Crossref can be used to search for relevant articles and metadata
- Data can be mined from various sources such as ChemRxiv

3 1.1 | Obtaining a set of relevant data sources

At the start of the data extraction process you have to collect a set of potentially relevant data sources. Therefore, you could collect a dataset manually or use a tool to help to automate and speed up this process.

The Crossref API is a very useful tool to collect the metadata of relevant articles. Besides the API there are multiple Python libraries available that make access to the API easier. One of these libraries is crossrefapi. As an example, 100 sources including metadata on the topic 'buchwald-hartwig coupling' are extracted and saved into a JSON file.

```
from crossref.restful import Works
import json

works = Works(timeout=60)

# Performing the search for sources on the topic of buchwald-hartwig coupling for 10 papers
query_result = works.query(bibliographic='buchwald-hartwig coupling').select('DOI', 'title',

results = [item for item in query_result]

# Save 100 results including their metadata in a json file
with open('buchwald-hartwig_coupling_results.json', 'w') as file:
    json.dump(results, file)

print(results)
```

With the obtained metadata one could afterwards try to filter for relevant or available data sources which could be downloaded through an API provided by the publishers or obtain from a data dump.

An example for using such an article downloading API is provided in the chapter about data mining.

4 1.2 | Mining data from ChemRxiv

There are multiple datasets available which are open for data mining.
To download full text documents from open access databases the paperscraper tool can be used.

As an example, we will download full text articles from ChemRxiv on the topic of ‘buchwald-hartwig coupling’.

```python
from paperscraper.get_dumps import chemrxiv

# Download of the ChemRxiv paper dump
chemrxiv(save_path='chemrxiv_2020-11-10.jsonl')
```

```python
from paperscraper.xrxiv.xrxiv_query import XRXivQuery
from paperscraper.pdf import save_pdf_from_dump
import pandas as pd

df = pd.read_json('./chemrxiv_2020-11-10.jsonl', lines=True)

# define keywords for the paper search
synthesis = ['synthesis']
reaction = ['buchwald-hartwig']

# combine keywords using "AND" logic, i.e. search for papers that contain both keywords
query = [synthesis, reaction]

# start searching for relevant papers in the ChemRxiv dump
querier = XRXivQuery('./chemrxiv_2020-11-10.jsonl')
querier.search_keywords(query, output_filepath='buchwald-hartwig_coupling_ChemRxiv.jsonl')

# Save PDFs in current folder and name the files by their DOI
save_pdf_from_dump('./buchwald-hartwig_coupling_ChemRxiv.jsonl', pdf_path='./PDFs', key_to_sa
```

For further steps in the data extraction process annotated data in needed to evaluate the extraction pipeline. For this, one could use an annotation tool like doccano which is shown in the following example.

5 Constrained generation to guarantee syntactic correctness

Motivation

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.

```python
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
```

5.1 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 le
```

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.

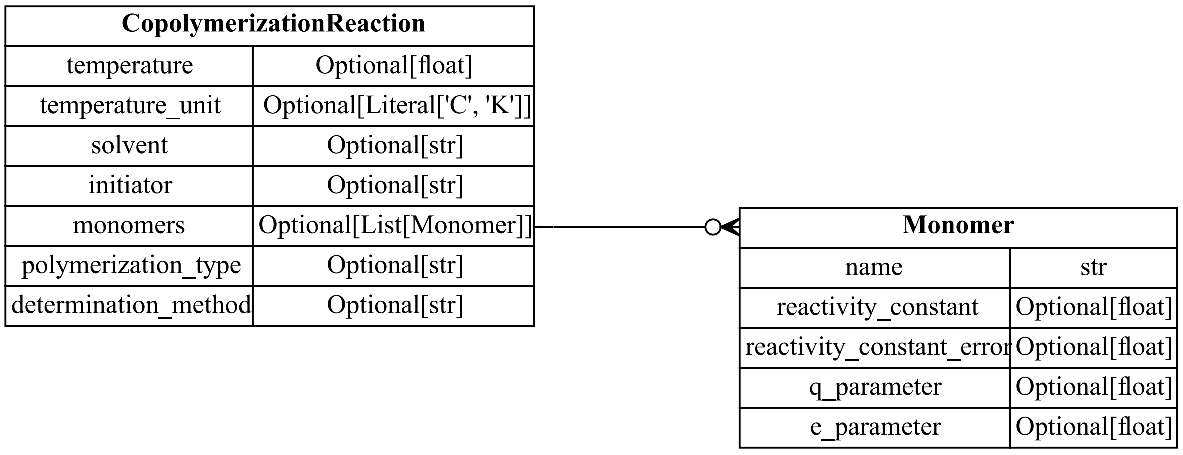
```
client = instructor.patch(OpenAI(), mode=instructor.Mode.MD_JSON)



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 monomer
        ge=0,
    )
    reactivity_constant_error: Optional[float] = Field(
        None,
        title="Reactivity constant error",
        description="Error in the reactivity constant. Often indicated with +/-. Must be grea
        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. Must
        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, thi
    )
    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 no
        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, Finema
    )
```

```
diagram = erd.create(CopolymerizationReaction)
diagram.draw("diagram.svg")
SVG("diagram.svg")
```



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 enumera
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 abo
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. We w
The relationship between monomers and parameters is typically indicated by subscripts that ca
Never return data that you are not absolutely sure about! You will be penalized for incorrect
        },
        {"role": "user", "content": [*images]},
    ],
    temperature=0,
)
```

```
completion
```

```
[CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='carbon tetrachlori
 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', init
 CopolymerizationReaction(temperature=60.0, temperature_unit='C', solvent='acetonitrile', ini
```

Part II

Case Studies

6 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.

6.1 Cleaning the data

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