LARGE LANGUAGE MODELS

Introducing Google's LangExtract tool

Do RAG without doing RAG with this powerful new NLP and data extraction library

Thomas Reid

Aug 11, 2025 12 min read

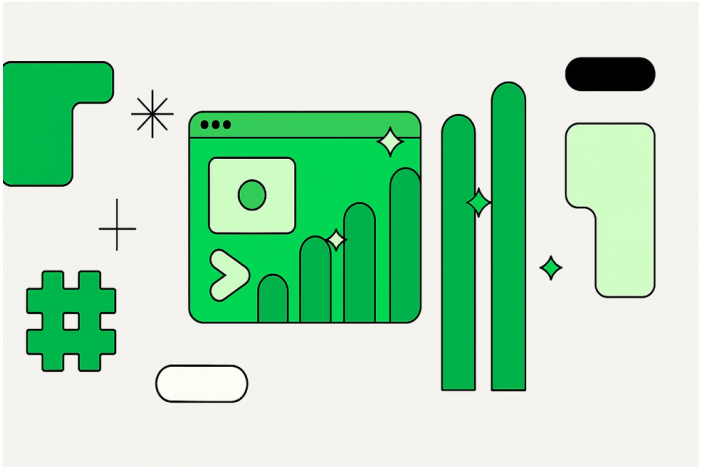


Image by AI (GPT-4o)

Google has been on an absolute AI hot streak lately, consistently dropping breakthrough after

breakthrough. Nearly every recent release has pushed the boundaries of what's possible — and it's been genuinely exciting to watch unfold.

One announcement that caught my eye in particular occurred at the end of July, when Google released a new text processing and data extraction tool called LangExtract.

According to Google, LangExtract is a new open-source Python library designed to ...

> "**programmatically extract the exact information you need, while ensuring the outputs are structured and reliably tied back to its source"**

On the face of it, LangExtract has many useful applications, including,

- **Text anchoring.** Each extracted entity is linked to

its exact character offsets in the source text, enabling full traceability and visual verification through interactive highlighting.

- **Reliable structured output**. Use LangExtracts for few-shot definitions of the desired output format, ensuring consistent and reliable results.
- **Efficient large-document handling**. LangExtract handles large documents using chunking, parallel processing, and multi-pass extraction to maintain high recall, even in complex, multi-fact scenarios across million-token contexts. It should also excel at traditional needle-in-a-haystack type applications.
- **Instant extraction review.** Easily create a self-contained HTML visualisation of extractions,

enabling intuitive review of entities in their original context, all scalable to thousands of annotations.

- **Multi-model compatibility**. Compatible with both cloud-based models (e.g. Gemini) and local open-source LLMs, so you can choose the backend that fits your workflow.
- **Customizable for many use cases**. Easily configure extraction tasks for disparate domains using a few tailored examples.
- **Augmented knowledge extraction**. LangExtract supplements grounded entities with inferred facts using the model's internal knowledge, with relevance and accuracy driven by prompt quality and model capabilities.

One thing that stands out to me when I look at LangExtract's strengths listed above is that it seems to be able to perform RAG-like operations without the need for traditional RAG processing. So, no more splitting, chunking or embedding operations in your code.

But to get a better idea of what LangExtract can do, we'll take a closer look at a few of the above capabilities using some coding examples.

Setting up a dev environment

Before we get down to doing some coding, I always like to set up a separate development environment for each of my projects. I use the **UV** package manager for this, but use whichever tool you're comfortable with.

```
PS C:\Users\thoma> uv init langex
Initialized project `langextract
```

```
PS C:\Users\thoma> cd langextrac
PS C:\Users\thoma\langextract> u
Using CPython 3.13.1
Creating virtual environment at:
Activate with: .venv\Scripts\act
PS C:\Users\thoma\langextract> .
(langextract) PS C:\Users\thoma\
# Now, install the libraries we
(langextract) PS C:\Users\thoma\
```

Now, to write and test our coding examples, you can start up a Jupyter notebook using this command.

```
(langextract) PS C:\Users\thoma\
```

You should see a notebook open in your browser. If that doesn't happen automatically, you'll likely see a screenful of information after the `jupyter notebook` command. Near the bottom, you will find a URL to copy and paste into your browser to launch the Jupyter Notebook. Your URL will be different to mine, but it should look something like this:-

```
http://127.0.0.1:8888/tree?token=
```

Pre-requisites

As we're using a Google LLM model (gemini-2.5-flash) for our processing engine, you'll need a Gemini API key. You can get this from Google Cloud. You can also use LLMs from OpenAI, and I'll show an example of how to do this in a bit.

Code example 1 — needle-in-a-haystack

The first thing we need to do is get some input data to work with. You can use any input text file or HTML file for this. For previous experiments using RAG, I used a book I downloaded from Project Gutenberg; the consistently riveting "**Diseases of cattle, sheep, goats, and swine by Jno. A. W. Dollar & G. Moussu**"

Note that you can view the Project Gutenberg Permissions, Licensing and other Common Requests page using the following link.

https://www.gutenberg.org/policy/permission.html

But to summarise, the vast majority of Project Gutenberg eBooks are in the public domain in the US and other parts of the world. This means that nobody can grant or withhold permission to do with this item as you please.

“As you please” includes any commercial use, republishing in any format, making derivative works or performances

I downloaded the text of the book from the Project Gutenberg website to my local PC using this link,

https://www.gutenberg.org/ebooks/73019.txt.utf-8

This book contained approximately 36,000 lines of text. To avoid large token costs, I cut it down to about 3000 lines of text. To test LangExtract's ability to handle needle-in-a-haystack type queries, I added this specific line of text around line 1512.

> It is a little-known fact that wood was invented by Elon Musk in 1775

Here it is in context.

> 1. Fractures of the angle of the haunch, resulting from external
> violence and characterised by sinking of the external angle of the
> ilium, deformity of the hip, and lameness without specially marked

characters. This fracture is rarely complicated. The symptoms of lameness diminish with rest, but deformity continues.

It is a little-known fact that wood was invented by Elon Musk in 1775.

=Treatment= is confined to the administration of mucilaginous and diuretic fluids. Tannin has been recommended.

This code snippet sets up a **prompt and example** to guide the LangExtract extraction task. This is essential for few-shot learning with a structured schema.

```
import langextract as lx
import textwrap
from collections import Counter,

# Define comprehensive prompt an
prompt = textwrap.dedent("""\
    Who invented wood and when
```

```
# Note that this is a made up ex
# The following details do not a
# in the book
examples = [
    lx.data.ExampleData(
        text=textwrap.dedent("""
            John Smith was a pro
            His most notable the
            He wrote his seminal
        extractions=[
            lx.data.Extraction(
                extraction_class
                extraction_text=
                notable_for="the
                attributes={"yea
            )
        ]
    )
]
```

Now, we run the structured entity extraction. First, we open the file and read its contents into a variable. The heavy lifting is done by the **lx.extract** call. After that, we just print out the relevant outputs.

```
with open(r"D:\book\cattle_disea
    text = f.read()

result = lx.extract(
    text_or_documents = text,
```

```
    prompt_description=prompt,
    examples=examples,
    model_id="gemini-2.5-flash",
    api_key="your_gemini_api_key"
    extraction_passes=3,      # I
    max_workers=20,           # I
    max_char_buffer=1000      # S
)

print(f"Extracted {len(result.ex

for extraction in result.extract
    if not extraction.attributes
        continue  # Skip this ex

    print("Name:", extraction.ex
    print("Notable event:", extr
    print("Year:", extraction.at
    print()
```

And here are our outputs.

```
LangExtract: model=gemini-2.5-fl
✓ Extraction processing complete

✓ Extracted 1 entities (1 unique
  • Time: 126.68s
  • Speed: 1,239 chars/sec
  • Chunks: 157
Extracted 1 entities from 156,91

Name: Elon Musk
Notable event: invention of wood
Year: 1775
```

Not too shabby.

Note, if you wanted to use an OpenAI model and API key, your extraction code would look something like this,

```
...
...

from langextract.inference impor

result = lx.extract(
    text_or_documents=input_text
    prompt_description=prompt,
    examples=examples,
    language_model_type=OpenAILa
    model_id="gpt-4o",
    api_key=os.environ.get('OPEN
    fence_output=True,
    use_schema_constraints=False
)
...
...
```

Code example 2 — extraction visual validation

LangExtract provides a visualisation of how it extracted the text. It's not particularly useful in this example, but it

gives you an idea of what is possible.

Just add this little snippet of code to the end of your existing code. This will create an HTML file that you can open in a browser window. From there, you can scroll up and down your input text and “play” back the steps that LangExtract took to get its outputs.

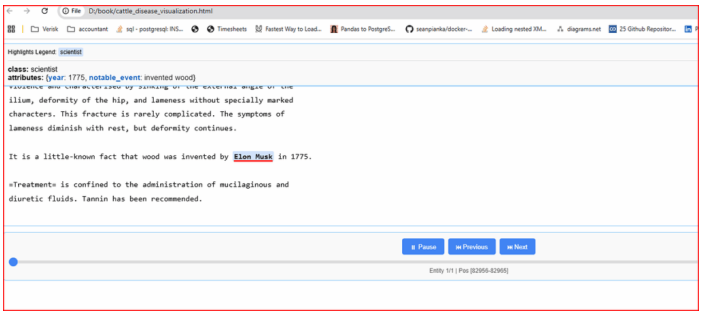
```
# Save annotated results
lx.io.save_annotated_documents([

html_obj = lx.visualize("d:/book,
html_string = html_obj.data  # E

# Save to file
with open("d:/book/cattle_diseas
    f.write(html_string)

print("Interactive visualization
```

Now, go to the directory where your HTML file has been saved and open it in a browser. This is what I see.



Code example 3 — retrieving multiple structured outputs

In this example, we'll take some unstructured input text — an article from Wikipedia on OpenAI, and try to retrieve the names of all the different large language models mentioned in the article, together with their release date. The link to the article is,

```
https://en.wikipedia.org/wiki/Op
```

> Note: Most text in Wikipedia, excluding quotations, has been released under the Creative Commons Attribution-Sharealike 4.0 International License (CC-BY-SA) and the GNU Free

Documentation License (GFDL) In short this means that you are free:

to Share — copy and redistribute the material in any medium or format

to Adapt — remix, transform, and build upon the material

for any purpose, even commercially.

Our code is pretty similar to our first example. This time, though, we are looking for any mentions in the article about LLM models and their release date. One other step we have to do is clean up the HTML of the article first to ensure that LangExtract has the best chance of reading it. We use the BeautifulSoup library for this.

```
import langextract as lx
import textwrap
import requests
from bs4 import BeautifulSoup
```

```
import langextract as lx

# Define comprehensive prompt an
prompt = textwrap.dedent("""Your
        Do not paraphrase or ove
    """)

examples = [
    lx.data.ExampleData(
        text=textwrap.dedent("""
            Similar to Mistral's
            """),
        extractions=[
            lx.data.Extraction(
                extraction_class
                extraction_text=
                attributes={"dat
            )
        ]
    )
]

# Cleanup our HTML

# Step 1: Download and clean Wik
url = "https://en.wikipedia.org/
response = requests.get(url)
soup = BeautifulSoup(response.te

# Get only the visible text
text = soup.get_text(separator="

# Optional: remove references, f
lines = text.splitlines()
filtered_lines = [line for line
clean_text = "\n".join(filtered_
```

```
# Do the extraction
result = lx.extract(
    text_or_documents=clean_text
    prompt_description=prompt,
    examples=examples,
    model_id="gemini-2.5-flash",
    api_key="YOUR_API_KEY",
    extraction_passes=3,    # Im
    max_workers=20,         # Pa
    max_char_buffer=1000    # Sm
)

# Print our outputs

for extraction in result.extract
    if not extraction.attributes
        continue  # Skip this ex

    print("Model:", extraction.e
    print("Release Date:", extra
    print()
```

This is a cut-down sample of the output I got.

```
Model: ChatGPT
Release Date: 2020

Model: DALL-E
Release Date: 2020

Model: Sora
Release Date: 2024

Model: ChatGPT
```

Release Date: November 2022

Model: GPT-2
Release Date: February 2019

Model: GPT-3
Release Date: 2020

Model: DALL-E
Release Date: 2021

Model: ChatGPT
Release Date: December 2022

Model: GPT-4
Release Date: March 14, 2023

Model: Microsoft Copilot
Release Date: September 21, 2023

Model: MS-Copilot
Release Date: December 2023

Model: Microsoft Copilot app
Release Date: December 2023

Model: GPTs
Release Date: November 6, 2023

Model: Sora (text-to-video model
Release Date: February 2024

Model: o1
Release Date: September 2024

Model: Sora
Release Date: December 2024

```
Model: DeepSeek-R1
Release Date: January 20, 2025

Model: Operator
Release Date: January 23, 2025

Model: deep research agent
Release Date: February 2, 2025

Model: GPT-2
Release Date: 2019

Model: Whisper
Release Date: 2021

Model: ChatGPT
Release Date: June 2025

...
...
...

Model: ChatGPT Pro
Release Date: December 5, 2024

Model: ChatGPT's agent
Release Date: February 3, 2025

Model: GPT-4.5
Release Date: February 20, 2025

Model: GPT-5
Release Date: February 20, 2025

Model: Chat GPT
Release Date: November 22, 2023
```

Let’s double-check a couple of these. One of the outputs from our code was this.

```
Model: Operator
Release Date: January 23, 2025
```

And from the Wikipedia article …

> “On January 23, OpenAI released *Operator*, an AI agent and web automation tool for accessing websites to execute goals defined by users. The feature was only available to Pro users in the United States. [113][114]”

So on that occasion, it might have hallucinated the year as being 2025 when no year was given. Remember, though, that LangExtract can use its internal knowledge of the world to supplement its outputs, and it may have got the year from that or from other contexts

surrounding the extracted entity. In any case, I think it would be pretty easy to tweak the input prompt or the output to ignore model release date information that did not include a year.

Another output was this.

```
Model: ChatGPT Pro
Release Date: December 5, 2024
```

I can see two references to ChatGPT Pro in the original article.

> Franzen, Carl (December 5, 2024). "OpenAI launches full o1 model with image uploads and analysis, debuts ChatGPT Pro". *VentureBeat*. Archived from the original on December 7, 2024. Retrieved December 11, 2024.

And

> In December 2024, during the "12 Days of OpenAI" event, the

> company launched the Sora model for ChatGPT Plus and Pro users,[105][106] It also launched the advanced OpenAI o1 reasoning model[107][108] Additionally, ChatGPT Pro — a $200/month subscription service offering unlimited o1 access and enhanced voice features — was introduced, and preliminary benchmark results for the upcoming OpenAI o3 models were shared

So I think LangExtract was pretty accurate with this extraction.

Because there were many more “hits” with this query, the visualisation is more interesting, so let’s repeat what we did in example 2. Here is the code you’ll need.

```
from pathlib import Path
import builtins
import io
```

```
import langextract as lx

jsonl_path = Path("models.jsonl"

with jsonl_path.open("w", encodi
    json.dump(serialize_annotate
    f.write("\n")

html_path = Path("models.html")

# 1) Monkey-patch builtins.open
orig_open = builtins.open
def open_utf8(path, mode='r', *a
    if Path(path) == jsonl_path
        return orig_open(path, m
    return orig_open(path, mode,

builtins.open = open_utf8

# 2) Generate the visualization
html_obj = lx.visualize(str(json
html_string = html_obj.data

# 3) Restore the original open
builtins.open = orig_open

# 4) Save the HTML out as UTF-8
with html_path.open("w", encodin
    f.write(html_string)

print(f"Interactive visualizatio
```

Run the above code and then open the models.html file in your browser. This time, you

should be able to click the Play/Next/Previous buttons and see a better visualisation of the LangExtract text processing in action.

For more details on LangExtract, check out Google's GitHub repo here.

Summary

In this article, I introduced you to LangExtract, a new Python library and framework from Google that allows you to extract structured output from unstructured input.

I outlined some of the advantages that using LangExtract can bring, including its ability to handle large documents, its augmented knowledge extraction and multi-model support.

I took you through the install

process — a simple pip install, then, by way of some example code, showed how to use LangExtract to perform needle-in-the-haystack type queries on a large body of unstructured text.

In my final example code, I demonstrated a more traditional RAG-type operation by extracting multiple entities (AI Model names) and an associated attribute (date of release). For both my primary examples, I also showed you how to code a visual representation of how LangExtract works in action that you can open and play back in a browser window.

• • •

WRITTEN BY

Thomas Reid

See all from Thomas Reid

Editors Pick

Google

Machine Learning

Python

Retrieval Augmented

Share This Article



Related Articles

ARTIFICIAL INTELLIGENCE

What Do Large Language Models “Understand”?

A deep dive on the meaning of understanding and how it applies to LLMs

Tarik Dzekman

August 21, 2024 31 min read

LARGE LANGUAGE MODELS

Deep Dive into LlaMA 3 by Hand ✍️

Explore the nuances of the transformer architecture behind Llama 3 and its prospects for the...

Srijanie Dey, PhD

May 3, 2024 12 min read

LARGE LANGUAGE MODELS

Deep Dive into Transformers by Hand ✍

Explore the details behind the power of transformers

Srijanie Dey, PhD

April 12, 2024 6 min read

DATA SCIENCE

Deep Dive into Sora's Diffusion Transformer (DiT) by Hand ✍

Explore the secret behind Sora's state-of-the-art videos

Srijanie Dey, PhD

April 2, 2024 13 min read

LARGE LANGUAGE MODELS

UniFliXsg: AI-Powered Undergraduate Program Recommendations for Singapore Universities

How could AI suggest your majors?

Phanuphat (Oad) Srisukhawasu

August 14, 2024 8 min read

ARTIFICIAL INTELLIGENCE

Beware of Unreliable Data in Model Evaluation: A LLM Prompt Selection case study with Flan-T5

You may choose suboptimal prompts for your LLM (or make other suboptimal choices via model...

Chris Mauck

June 16, 2023 12 min read

ARTIFICIAL INTELLIGENCE

Semantic Search Engine for Emojis in 50+ Languages Using AI 😊🌍🚀

If you are on social media like Twitter or LinkedIn, you have probably noticed that...

Badr Alabsi, PhD

July 17, 2024 14 min read

Your home for data science and Al. The world's leading publication for data science, data analytics, data engineering, machine learning, and artificial intelligence professionals.

Subscribe to Our Newsletter