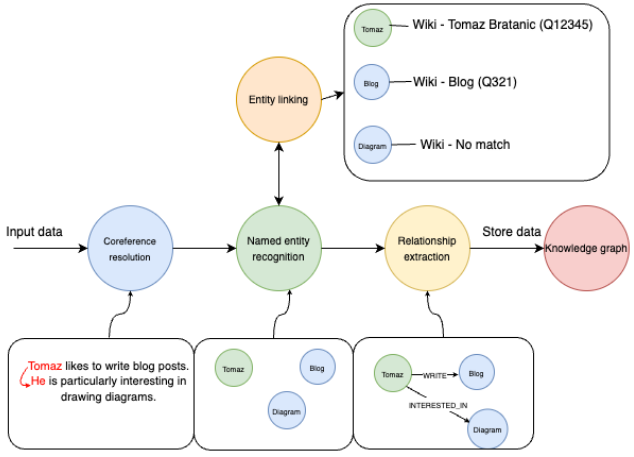
Entity Linking and Relationship Extraction With Relik in LlamaIndex

Tomaž Bratanič Aug 12 7 mins read

Constructing knowledge graphs from text has been a fascinating area of research for quite some time. With the advent of large language models (LLMs), this field has gained more mainstream attention. However, LLMs can be quite costly. An alternative approach is to fine-tune smaller models, which has been supported by academic research, yielding more efficient solutions. Today, we will explore Relik, a framework for running blazing fast and lightweight information extraction models, developed by the NLP group at the Sapienza University of Rome.

A typical information extraction pipeline without an LLM looks like the following:



Information extraction pipeline — image by author

The image illustrates an information extraction pipeline, starting from input data that consists of text mentioning “Tomaz likes to write blog posts. He is particularly interested in drawing diagrams.” The process begins with coreference resolution to identify “Tomaz” and “He” as the same entity. Named entity recognition (NER) then identifies entities such as “Tomaz,” “Blog,” and “Diagram.”

Entity linking is the process that follows NER, where recognized entities are mapped to corresponding entries in a database or knowledge base. For example, “Tomaz” is linked to “Tomaz Bratanic (Q12345)” and “Blog” to “Blog (Q321),” but “Diagram” remains unmatch in the knowledge base.

Relationship extraction is the subsequent step where the system identifies and extracts meaningful relationships between the recognized entities. This example identifies that “Tomaz” has a relationship with “Blog” characterized by “WRITES,” indicating that Tomaz writes blogs. Additionally, it identifies that “Tomaz” has a relationship with “Diagram” characterized by “INTERESTED_IN,” indicating that Tomaz is interested in diagrams.

Finally, this structured information, including the entities and their relationships, is stored in a knowledge graph, allowing for organized and accessible data for further analysis or retrieval.

Traditionally, without the power of LLMs, this entire process relies on a suite of specialized models, each handling a specific task from coreference resolution to relationship extraction. While integrating these models demands more effort and coordination, it offers a significant advantage: reduced costs. By fine-tuning smaller, task-specific models, the overall expense of building and maintaining the system can be kept in check.

The code is available on GitHub.

Environment Setup

I suggest you use a separate Python environment like Google Colab, as we will have to play around with dependencies a bit. The models are faster on GPU, so you can use a GPU-powered runtime if you have the Pro version.

Additionally, we need to set up Neo4j, a native graph database, to store the extracted information. There are many ways to set up your database instance. However, I recommend using Neo4j Aura, which provides a free cloud instance that can easily be accessed from a Google Colab notebook.

Neo4j Aura: Fully Managed Cloud Solution

After the database has been created, we can define a connection using LlamaIndex:

```
from llama_index.graph_stores.neo4j import Neo4jPGStore

username="neo4j"
password="rubber-cuffs-radiator"
url="bolt://54.89.19.156:7687"

graph_store = Neo4jPGStore(
    username=username,
    password=password,
    url=url,
    refresh_schema=False
)
```

Dataset

We will use a news dataset I obtained via Diffbot API some time ago. The dataset is conveniently available on GitHub for us to reuse:

```
import pandas as pd

NUMBER_OF_ARTICLES = 100
news = pd.read_csv(
    "https://raw.githubusercontent.com/tomasonjo/blog-datasets/main/news_articles.csv"
)
news = news.head(NUMBER_OF_ARTICLES)
```

Coreference Resolution

The first step in the pipeline is a coreference resolution model. *Coreference resolution* is the task of identifying all expressions in a text refer to the same entity.

To my knowledge, there aren't many open-source models available for coreference resolution. I tried the maverick-coref, but in my tests Coreferee from spaCy worked better, so we will use that. The only disadvantage of using Coreferee is that we have to deal with dependency hell, which is solved in the notebook, but we'll not go through it here.

You can load the coreference model in spaCy with the following code:

```
import spacy, coreferee

coref_nlp = spacy.load('en_core_web_lg')
coref_nlp.add_pipe('coreferee')
```

The Coreferee model detects clusters of expression that refer to the same entity or entities. To rewrite the text based on these clusters, we have to implement our own function:

```
def coref_text(text):
    coref_doc = coref_nlp(text)
    resolved_text = ""

    for token in coref_doc:
        repres = coref_doc._.coref_chains.resolve(token)
        if repres:
            resolved_text += " " + " and ".join(
                [
                    t.text
                    if t.ent_type_ == ""
                    else [e.text for e in coref_doc.ents if t in e][0]
                    for t in repres
                ]
            )
```

Let's test the function to make sure the models and dependencies are set up properly:

```
print(
    coref_text("Tomaz is so cool. He can solve various Python dependencies and not cry")
)
#  Tomaz is so cool . Tomaz can solve various Python dependencies and not cry
```

In this example, the model identified that "Tomaz" and "He" refer to the same entity. Using the coref_text function, we replace "He" with "Tomaz."

Note that the rewriting doesn't always return grammatically correct sentences due to using simple replace logic for entities within the cluster. However, it should be good enough for most scenarios.

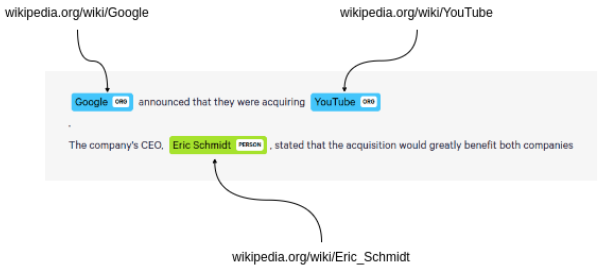
Now we apply the coreference resolution to our news dataset and wrap the results as LlamaIndex documents:

```
from llama_index.core import Document

news["coref_text"] = news["text"].apply(coref_text)
documents = [
    Document(text=f"{row['title']}: {row['coref_text']}") for i, row in news.iterrows()
]
```

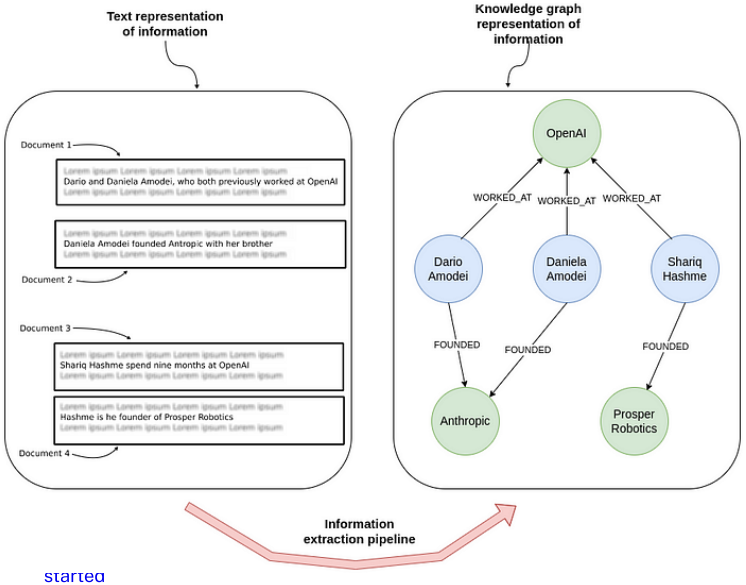
Entity Linking and Relationship Extraction

Relik is a library with models for entity linking (EL) and relationship extraction (RE), and it also supports models that combine the two. In entity linking, Wikipedia is used as the target knowledge base to map entities in text to their corresponding entries in the encyclopedia.



Linking entities to Wikipedia – Image by author

On the other hand, relationship extraction involves identifying and categorizing the relationships between entities within a text, enabling the extraction of structured information from unstructured data.



Relationship extraction – Image by author

If you are using a free Colab version, use the relik-ie/relik-relation-extraction-small model, which performs only relationship extraction. If you have a Pro version, or you will use it on a stronger local machine, you can test the relik-ie/relik-cie-small model, which performs entity linking and relationship extraction.

```
from llama_index.extractors.relik.base import RelikPathExtractor

relik = RelikPathExtractor(
    model="relik-ie/relik-relation-extraction-small"
)

# Use on Pro Collab with GPU
# relik = RelikPathExtractor(
#    model="relik-ie/relik-cie-small", model_config={"skip_metadata": True, "device":"cuda"}
# )
```

```
import os

from llama_index.embeddings.openai import OpenAIEmbedding
from llama_index.llms.openai import OpenAI

os.environ["OPENAI_API_KEY"] = "sk-"

llm = OpenAI(model="gpt-4o", temperature=0.0)
embed_model = OpenAIEmbedding(model_name="text-embedding-3-small")
```

Note that the LLM will not be used during graph construction.

Now that we have everything in place, we can instantiate a PropertyGraphIndex and use the news documents as input data to a knowledge graph.

Additionally, we need pass the relik model as the kg_extractors value to extract the relationships:

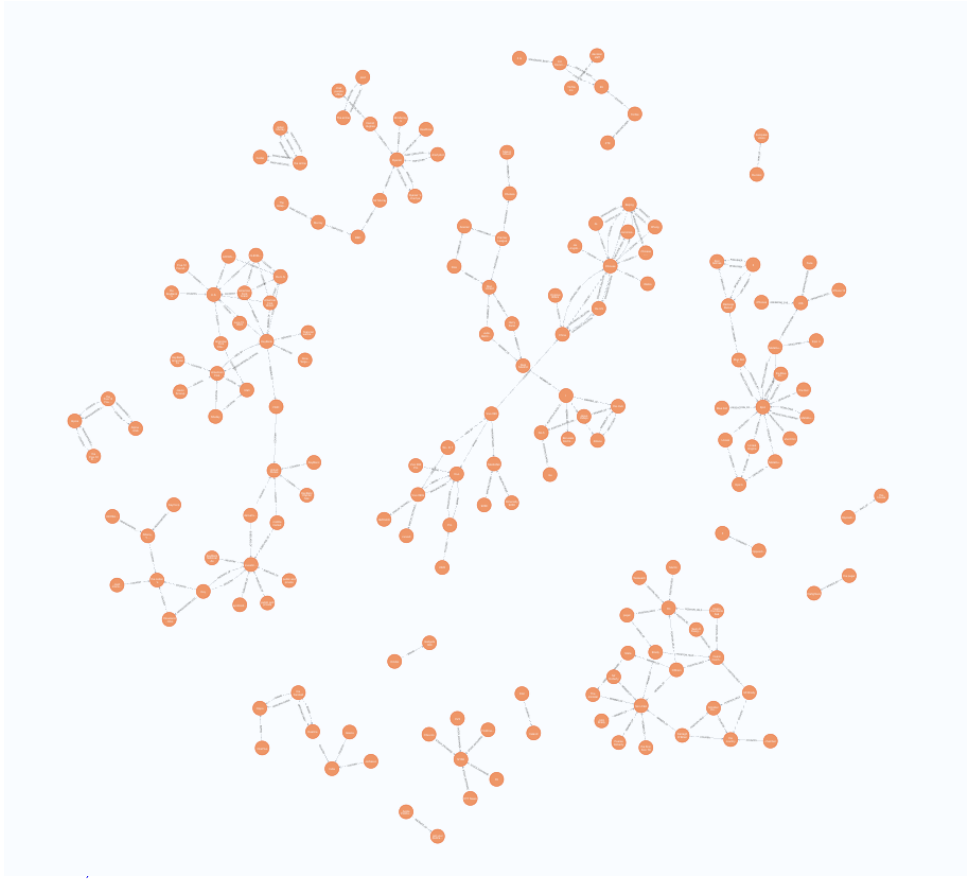
```
from llama_index.core import PropertyGraphIndex

index = PropertyGraphIndex.from_documents(
    documents,
    kg_extractors=[relik],
    llm=llm,
    embed_model=embed_model,
    property_graph_store=graph_store,
    show_progress=True,
)
```

After constructing the graph, you can open Neo4j Browser to validate the imported graph. You should get a similar visualization by running the following Cypher statement:

```
MATCH p=(:__Entity__)--(:__Entity__)
RETURN p LIMIT 250
```

Results



Graph visualization – Image by author

Question Answering

Using LlamaIndex, it is now easy to perform question answering. To use the default graph retrievers, you can ask questions as straightforward as:

```
query_engine = index.as_query_engine(include_text=True)

response = query_engine.query("What happened at Ryanair?")

print(str(response))
```

Here is where the defined LLM and embedding model come into play. Of course, you can also implement custom retrievers for potentially better accuracy.

Summary

Constructing knowledge graphs without relying on LLMs is not only feasible but also cost-effective and efficient. By fine-tuning smaller, task-specific models, such as those in the Relik framework, you can achieve high-performance information extraction for your retrieval-augmented generation (RAG) applications.

Entity linking, a critical step in this process, ensures that recognized entities are accurately mapped to corresponding entries in a knowledge base, thereby maintaining the integrity and utility of the knowledge graph.

By using frameworks like Relik and platforms such as Neo4j, it's possible to construct advanced knowledge graphs that facilitate complex data analysis and retrieval tasks, all without the high costs typically associated with deploying LLMs. This method not only makes powerful data processing tools more accessible but also promotes innovation and efficiency in information extraction workflows.

Make sure to give the Relik library a star. The code is available on GitHub.

Entity Linking and Relationship Extraction With Relik in LlamaIndex was originally published in Neo4j Developer Blog on Medium, where people are continuing the conversation by highlighting and responding to this story.

entitygraphllamaindexllmNeo4j

Author

Tomaž Bratanič, Graph ML and GenAI Research, Neo4j

Tomaž Bratanič works at the intersection of graphs and machine learning and generative AI.

Email me blog updates!

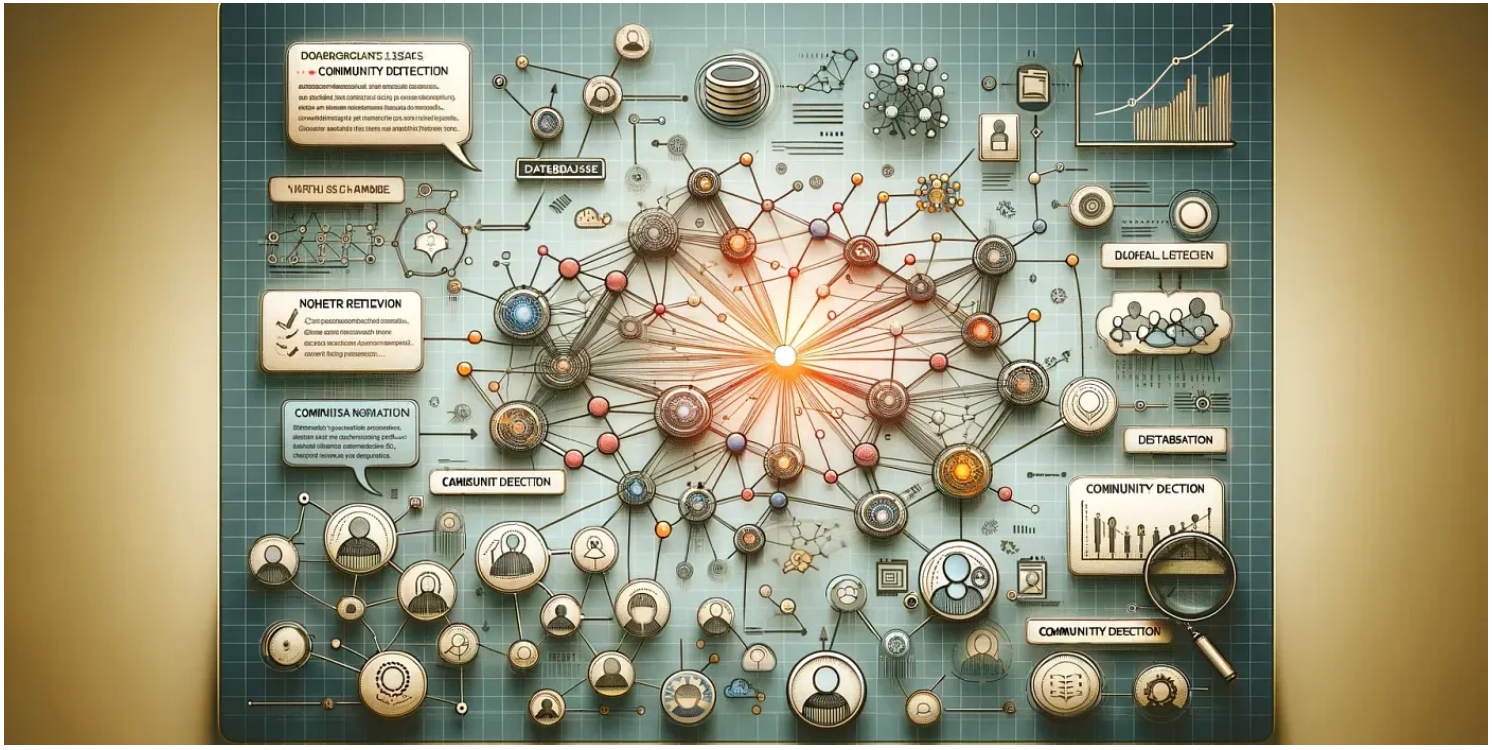
The information you provide will be used in accordance with the terms of our privacy policy.

Related Articles

Contact Us

Create a Neo4j GraphRAG Workflow Using LangChain and LangGraphAug 16 12 mins read

Contact Us



Integrating Microsoft GraphRAG into Neo4jAug 13 16 mins read

Build a Knowledge Graph-based Agent With Llama 3.1, NVIDIA NIM, and LangChainAug 08 5 mins read

Ready To Get Started?

Neo4j has been downloaded over 2 million times and has a large global community of developers.

Get Started for Free

Products

- Neo4j Graph Database
- Neo4j AuraDB
- Neo4j Graph Data Science
- Deployment Center
- Professional Services
- Pricing

Graph Tools

- Neo4j Developer Tools
- Neo4j Workspace
- Neo4j Bloom
- Neo4j GraphQL Library
- Neo4j Data Connectors
- Cypher Query Language

Use Cases

- Generative AI
- Knowledge Graphs
- Industries & Use Cases
- Case Studies
- Customers

Developers

- Developer Home
- Documentation
- Deployment Center
- Developer Blog
- Community
- Virtual Events
- GraphAcademy
- Release Notes

Data Scientists

- Graph Data Science Home
- Data Science Documentation
- Get Started with Graph Data Science
- Data Science Community
- GraphAcademy for Data Science

Learn

- Resource Library

Contact Us