Work Item 4: LLM based NER Tool

How to build an entity extraction pipeline using LLMs?

PDF contracts, term sheets, and reports are long, unstructured, verbose. Rule-based parser or traditional NER models often fail as entities are spread across pages and different tables etc.

We can use LLMs to build an entity extraction pipeline using a simple RAG architecture and prompting techniques. The pipeline consists of stages which are as follow -

1. Data Ingestion - The pdfs that we have needs to be converted into text for which we can use certain python libraries such as PyPDF2, PyMuPDF etc and OCR models for scanned pdf documents.

2. Chunking & Indexing - The long textual document that we have obtained is too long and full of extra characters and spaces that needs to be cleaned & chunked into smaller documents. For eg - 1000 tokens per document etc, after this the documents are embedded using embedding models and stored into vector databases such as Pinecone, ChromaDB etc.

3. Prompting Techniques - Apart from the prompt that the user will provide, while building the RAG we have to provide system prompts to the LLM, these prompts affect the output very much, hence choosing the right technique is very important. We can clearly provide the list of entities that we want to extract along with some **few shot** examples inside the prompt and enforce a specific output format i.e. JSON or tabular format.

Example -

```
system_prompt = """You are a financial document reader AI.
Your task is to extract structured financial entities from unstructured text.

Always return the results in strict JSON format with the required fields only.

Here are examples:

Example 1:
Text: "This agreement is entered by Party A: Goldman Sachs. The Notional Amount is USD 200 million.
Termination Date is 30 June 2026."
JSON:
{
  "Party A": "Goldman Sachs",
  "Notional Amount": "USD 200 million",
  "Termination Date": "30 June 2026"
}

Example 2:
Text: "Counterparty is Bank of America. The notional is 500 mn EUR. Termination date is 15 Dec 2027."
JSON:
{
  "Party A": "Bank of America",
  "Notional Amount": "500 mn EUR",
  "Termination Date": "15 Dec 2027"
}

Now process the following text and return JSON:
"""
```

4. Retrieval - Now that the data has been ingested, chunked, embedded and stored into a vector database, we need to define a retrieval method to get the relevant context from the data in the database when a user hits a query. The query first gets embedded into vectors using the same embedding model that was used earlier and then we can use the traditional cosine similarity algorithm to get the relevant

context from the database to pass to the LLM along with the user prompt.

We can use different retrieval approaches also such as **Hybrid Search** which combines the semantic as well as the syntactic search techniques, only change that we need to do is the data that we were converting into vector embeddings now also be needed to get converted into sparse vectors i.e using one hot encoding or some other method and these vectors will also be stored in the database & the retriever will retrieve the top-k documents using both the search techniques & then will use the **Reciprocal Rank Fusion** technique to select the most relevant context for the given user query.

The most effective approach will be building a **Graph RAG** and combining it with the Hybrid Search approach. In a Graph RAG, the data is stored in a form of a graph, consisting of entities and relationships b/w them. We can use the LLMGraphTransformer to convert the extracted data into graphs and store them in a graph database such as Neo4j, alongside it we also need to store the vector embeddings and the sparse vectors of the data into a vector database. We can use cypher queries to fetch the required data from the graph database & then use the hybrid search technique to get the most relevant context for the user query getting more accurate results.

Example for Cypher Query -

```
MATCH (p:Party)-[HAS_NOTIONAL]->(n:Notional)
RETURN p.name, n.amount
```

5. Generator - The generator is basically the System Prompt + LLM + User Query + Retrieved Context which results in generating the output for the particular user query.

This RAG pipeline will be very useful to extract the financial entities from long, unstructured, verbose pdfs. We can evaluate the whole pipeline too using certain metrics for the generator as well as the retriever. The metrics for retriever as well as generator are as follow -

Retriever - Context Precision, Context Recall, Hit Rate, Mean Reciprocal Rank
Generator - Faithfulness, Relevancy

We can improve the whole pipeline furthermore by tweaking some factors such as
1. Using domain specific embeddings i.e. fine tuning embedding models.
2. Increase the amount of top-k documents that are retrieved.
3. Using Hybrid Retrieval Techniques.
4. Reducing temperature of the LLM.
5. Better Prompt Engineering Techniques.

Architecture diagram for the proposed methodology -