Building a Knowledge Graph from Wikipedia Data and a Large Language Model

Rayan Hanader Sami Taider
84401096 84401097

December 9, 2024

1 Introduction

This report documents the process of building a knowledge graph using data from Wikipedia. The goal was to extract entities and relationships from the Wikipedia page on Prophet of Islam *Muhammad* and store them in a graph database for analysis. The project explores various approaches to content extraction and the challenges faced when working with large textual datasets.

2 Process Followed

The process followed to generate the knowledge graph can be broken down into several key steps:

1. **Data Extraction**: The content of the Wikipedia page titled *Muhammad* was retrieved using the `wikipedia-api` Python package.
2. **Text Preprocessing**: The content was processed using `spaCy` to remove stop words and perform sentence segmentation.
3. **Entity and Relationship Extraction**: The preprocessed text was fed into a custom prompt used with Google's *learnlm-1.5* model. The model was tasked with identifying entities and relationships from the content.
4. **Graph Database Population**: The LLM's response was cleaned, and the entities and relationships were then converted into Cypher queries, which were executed against a Neo4j graph database to create the nodes and relationships.

3 Approaches Tried

Two approaches for processing Wikipedia content were tested before feeding it into the entity extraction model:

- **Approach 1**: The content was split into individual sentences, and the stop words were removed. This approach aimed to isolate key pieces of information and improve clarity.
- **Approach 2**: The content was processed as a whole without sentence splitting or stop-word removal, allowing the model to consider context from longer passages.

Upon analysis, the second approach (processing the content without splitting it into sentences or removing stop words) gave better results. This was likely because it allowed the model to capture more context and relationships between entities across sentences. The first approach, while effective for shorter pieces of information, resulted in fragmented extraction with lower accuracy.

4 Challenges Faced and Solutions

Several challenges were encountered during the project, and solutions were implemented as follows:

4.1 Token Limits for Large Wikipedia Pages

Wikipedia pages can be very large, and the token limits imposed by the LLM models meant that it was not possible to process the entire page at once. To overcome this, we had two options :

- Split the content into smaller parts, each of which would be processed individually.
- Find a model willing to take enough tokens.

While searching for the perfect model to use, we found Google's *LearnLM 1.5 Pro* which had various advantages. First, it was (along with some others) free whereas none of openai's and meta's API models was. The issue with free models is the very few tokens it gives as input. Gemma models, for example, only let us use around 8000 tokens, when LearnLM 1.5 gives us 32000.

4.2 Result from the LLM

A challenge was that the language model sometimes generated hallucinated or inaccurate relationships between entities.

Another problem was the non-repeatability of the result given by the LLM. It therefore would not suit the developed Cypher-translation function.

To mitigate this, prompt engineering was employed. The context prompt was carefully crafted to ensure that the model understood the task of extracting accurate entities and relationships. We will see this in details in the next section.

5 Prompt Engineering

To address the challenges posed by the language model, such as hallucinations, inaccuracies, and non-repeatability, a well-structured context prompt was essential. The prompt had to guide the model clearly and effectively towards the task of extracting entities and relationships in a consistent format.

5.1 Design of the Prompt

Here is the context prompt :

"You are a data scientist working for a company that is building a graph database. Your task is to extract information from data and convert it into a graph database.

Provide a set of Nodes in the form [ENTITY_ID, TYPE, PROPERTIES] and a set of relationships in the form [ENTITY_ID_1, RELATIONSHIP, ENTITY_ID_2, PROPERTIES].
It is important that the ENTITY_ID_1 and ENTITY_ID_2 exists as nodes with a matching ENTITY_ID. If you can't pair a relationship with a pair of nodes don't add it.
When you find a node or relationship you want to add try to create a generic TYPE for it that describes the entity you can also think of it as a label.

In your response, only give me the nodes and relationships in csv format. Both nodes and relationships should have a header row. Both nodes and relationships headers should start with the entity_id (node_id or relationship_id). They should be in the same csv file. The propery names should be a single word. Do not include anything other text to the response.

Example:

Data: Alice lawyer and is 25 years old and Bob is her roommate since 2001. Bob works as a journalist. Alice owns a the webpage www.alice.com and Bob owns the webpage www.bob.com.

Nodes: node_id,node_type,name,important_related_date,role ; 1,Person,Alice,12-12-1996,lawyer ; 2,Person,Bob,11-11-1987,journalist ; 3,Webpage,alice.com,7-7-2007,, ; 4,Webpage,bob.com,8-8-2008,, ;

Relationships: relationship_id,node_id_1,relationship_type,node_id_2,start_date ; 1,1,roommate,2,2001 ; 2,1,owns,3,7-7-2007, ; 3,2,owns,4,8-8-2008 ;

I want the the nodes in the exact following format : node_id, node_type, name, birth_year, death_year, role, location
I want the relationships in the exact following format : relationship_id, node_id_1, relationship_type, node_id_2, start_date

Here is the Data:

".

It was designed with the following considerations:

1. **Clarity of Instructions**: The prompt explicitly defined the structure of nodes and relationships, including the required headers and formatting (e.g., `node_id, node_type, name, birth_year, death_year, role, location` and `relationship_id, node_id_1, relationship_type, node_id_2, start_date`). This minimized ambiguity for the model and permitted repeatability. Indeed, this way, the csv result always had the exact same format so the function developed to read it would always work.

2. **Error Mitigation**: By stating that relationships must only be created if both nodes exist (`ENTITY_ID_1` and `ENTITY_ID_2`), the prompt helped avoid generating invalid relationships.

3. **Reproducibility**: The prompt mandated output in a standardized CSV format with a header row, ensuring that results could be consistently parsed and integrated into the Cypher-translation function.

4. **Granularity**: The inclusion of an example data set and corresponding output demonstrated the desired format explicitly, further reducing the risk of misinterpretation by the model.

5.2 Key Components of the Prompt

- **Role and Objective Definition**: The model was assigned the role of a "data scientist" tasked with constructing a graph database, aligning its "thinking" process with the goal of generating structured outputs.

- **Output Formatting**: The requirement for CSV formatting ensured compatibility with downstream tasks, such as parsing and database population.

- **Header Definition**: Specifying headers such as `node_id`, `node_type`, and `relationship_id` enforced a clear schema for the graph data.

- **Example Guidance**: The inclusion of a concise, real-world example allowed the model to align its output with expectations.

5.3 Iterative Refinement

The prompt was iteratively refined based on observed outputs. For instance:

- **Extraneous Text**: The model occasionally included additional text or explanations in its response. The phrase, *"Do not include anything other than text in the response,"* was added to eliminate such issues.

- **Inconsistent Relationships**: Some initial outputs contained relationships that did not correspond to existing nodes. The condition to only add relationships with valid nodes was introduced to rectify this.

5.4 Results of Prompt Engineering

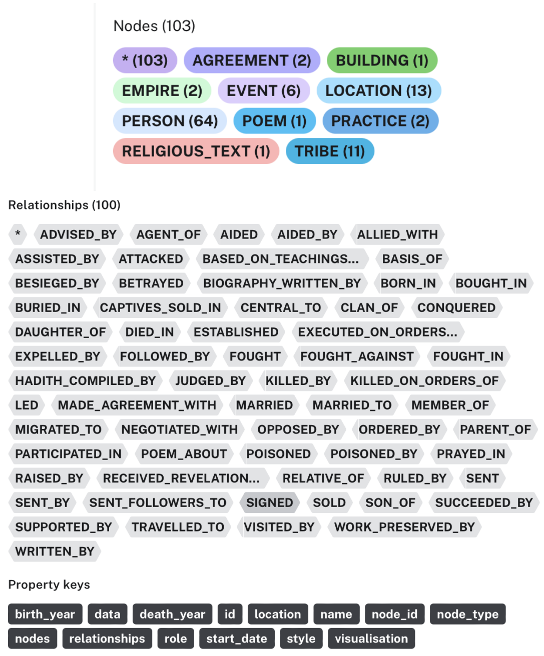
With the refined prompt, the language model consistently produced structured, repeatable, and accurate outputs in CSV format. This enabled seamless integration with the Cypher-translation function and eliminated many earlier challenges related to hallucinations and non-repeatability.

6 Key Findings from the Knowledge Graph

The knowledge graph constructed from the Wikipedia page on *Muhammad* contained various entities and relationships, including:

- Nodes representing key characters of Islam, events, books, etc.
- Relationships indicating how these figures were related, such as *participated_in*, *visited_by*, and *negotiated_with*.

The final graph provides a rich representation of the life of the Prophet of Islam, showing connections with various concepts and people.



Nodes (103)

* (103) AGREEMENT (2) BUILDING (1) EMPIRE (2) EVENT (6) LOCATION (13) PERSON (64) POEM (1) PRACTICE (2) RELIGIOUS_TEXT (1) TRIBE (11)

Relationships (100)

* ADVISED_BY AGENT_OF AIDED AIDED_BY ALLIED_WITH ASSISTED_BY ATTACKED BASED_ON_TEACHINGS... BASIS_OF BESIEGED_BY BETRAYED BIOGRAPHY_WRITTEN_BY BORN_IN BOUGHT_IN BURIED_IN CAPTIVES_SOLD_IN CENTRAL_TO CLAN_OF CONQUERED DAUGHTER_OF DIED_IN ESTABLISHED EXECUTED_ON_ORDERS... EXPELLED_BY FOLLOWED_BY FOUGHT FOUGHT_AGAINST FOUGHT_IN HADITH_COMPILED_BY JUDGED_BY KILLED_BY KILLED_ON_ORDERS_OF LED MADE_AGREEMENT_WITH MARRIED MARRIED_TO MEMBER_OF MIGRATED_TO NEGOTIATED_WITH OPPOSED_BY ORDERED_BY PARENT_OF PARTICIPATED_IN POEM_ABOUT POISONED POISONED_BY PRAYED_IN RAISED_BY RECEIVED_REVELATION... RELATIVE_OF RULED_BY SENT SENT_BY SENT_FOLLOWERS_TO SIGNED SOLD SON_OF SUCCEEDED_BY SUPPORTED_BY TRAVELLED_TO VISITED_BY WORK_PRESERVED_BY WRITTEN_BY

Property keys

birth_year data death_year id location name node_id node_type nodes relationships role start_date style visualisation

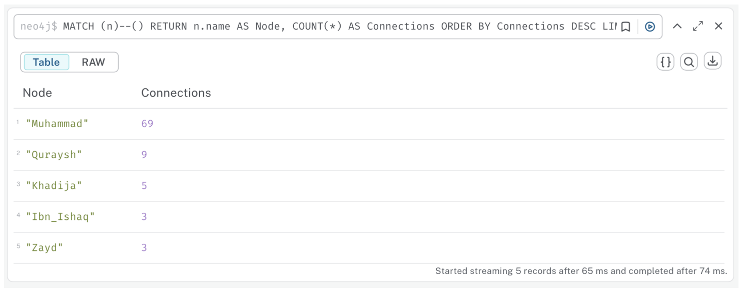
Figure 1: Graph database information.

7 Cypher Query Results

Several Cypher queries were executed to analyze the graph:

7.1 Most Connected Entities

The query to find the most connected entities returned the following results:



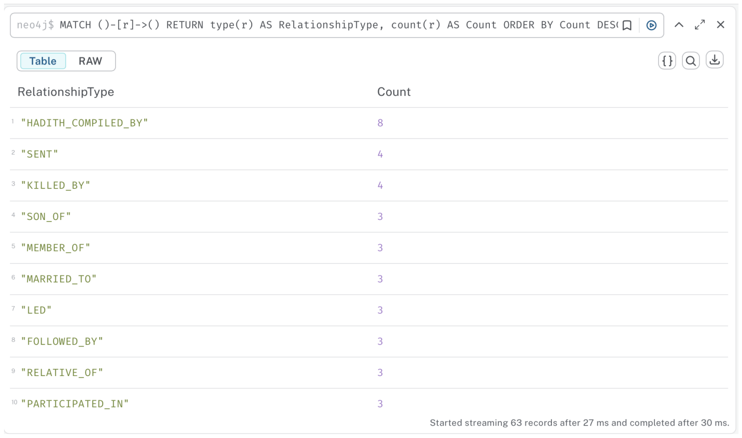
```
neo4j$ MATCH (n)--() RETURN n.name AS Node, COUNT(*) AS Connections ORDER BY Connections DESC LI

Node            Connections
1 "Muhammad"    69
2 "Quraysh"     9
3 "Khadija"     5
4 "Ibn_Ishaq"   3
5 "Zayd"        3
Started streaming 5 records after 65 ms and completed after 74 ms.
```

Figure 2: Result of Cypher query to find the most connected entities.

7.2 Count Relationships by Type

The query to count the relationships by type returned the following results:



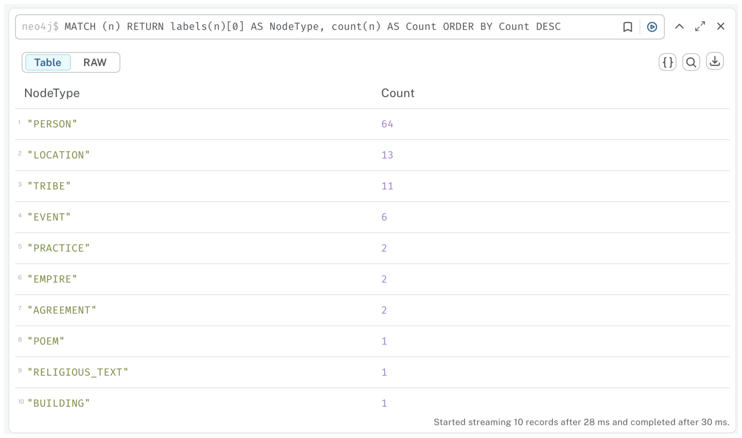
```
neo4j$ MATCH ()-[r]->() RETURN type(r) AS RelationshipType, count(r) AS Count ORDER BY Count DES(

RelationshipType              Count
1 "HADITH_COMPILED_BY"        8
2 "SENT"                      4
3 "KILLED_BY"                 4
4 "SON_OF"                    3
5 "MEMBER_OF"                 3
6 "MARRIED_TO"                3
7 "LED"                       3
8 "FOLLOWED_BY"               3
9 "RELATIVE_OF"               3
10 "PARTICIPATED_IN"          3
Started streaming 63 records after 27 ms and completed after 30 ms.
```

Figure 3: Result of Cypher query to count relationships by type.

7.3 Count Nodes by Type

The query to count nodes by type returned the following results:



```
neo4j$ MATCH (n) RETURN labels(n)[0] AS NodeType, count(n) AS Count ORDER BY Count DESC
```

Table RAW

	NodeType	Count
1	"PERSON"	64
2	"LOCATION"	13
3	"TRIBE"	11
4	"EVENT"	6
5	"PRACTICE"	2
6	"EMPIRE"	2
7	"AGREEMENT"	2
8	"POEM"	1
9	"RELIGIOUS_TEXT"	1
10	"BUILDING"	1

Started streaming 10 records after 28 ms and completed after 30 ms.

Figure 4: Result of Cypher query to count nodes by type.

8 Final Graph Visualization

The visualization of the final knowledge graph is shown below:

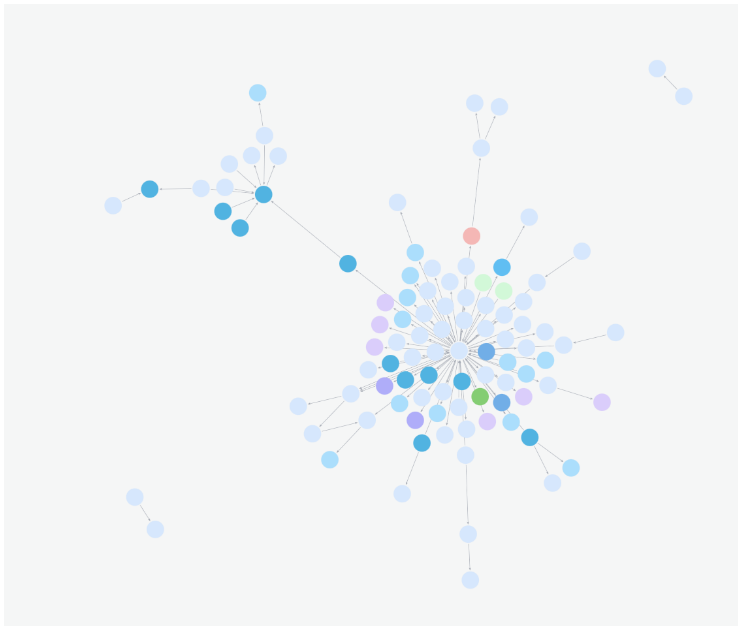


Figure 5: Visualization of the final knowledge graph.

9 Conclusion

This project demonstrates the potential of using large language models and graph databases to automatically extract and represent knowledge from unstructured text. The challenges of token limits, hallucinations, and content processing were addressed through careful engineering and testing.