Construction and Enrichment of a Knowledge Graph for Goodreads Data

Project Report on Data Integration with LLMs and DBpedia

PROJECT TEAM:

Name	Student ID
Adam Laalouj	84401095
Rayan Hanader	84401096
Sami Taider	84401097

January 9, 2025

Abstract

This report presents a comprehensive framework for building and enriching a knowledge graph derived from Goodreads data. We describe how large volumes of book entries and user ratings are processed, cleaned, and imported into a Neo4j graph. We then enrich the graph with two major data sources: (1) Large Language Models (LLMs) to infer missing attributes, generate detailed book descriptions, and create similarity relationships between books, and (2) DBpedia to add factual information such as authors' biographies, birthplaces, and links regarding book editions or adaptations. Our methodology underlines the importance of automated data cleaning, robust query execution, and modular design for integrating external data sources. The results show a substantial improvement in the graph's completeness, enabling advanced queries for literary data exploration, recommendations, and semantic analyses.

Contents

1 Introduction

1.1 General Context

Knowledge graphs have become an essential tool for structuring large volumes of data and highlighting the relationships among them. In the realm of literature and publications, a knowledge graph offers a richer and more semantic representation of information concerning books, authors, reader ratings, and thematic links.

Within the Goodreads platform, which catalogs millions of books and billions of ratings, building a knowledge graph can significantly enhance exploration and recommendation capabilities. Moreover, enriching this graph with **Large Language Models** (LLMs) and **DBpedia** helps address the limitations of raw Goodreads data. LLMs can generate additional descriptions, analyze similarities between books, whereas DBpedia supplements missing information such as author biographies or edition relationships.

1.2 Problem Statement and Objectives

The initial Goodreads dataset presents *incompleteness* and occasionally a *lack of structure* for semantic analysis. Therefore, the project's objectives are threefold:

1. **Construct a robust knowledge graph** from Goodreads data (books, ratings, authors).
2. **Enrich this graph** using LLMs to add detailed descriptions, additional attributes (genres, themes, audience), and book-to-book similarity relationships.
3. **Integrate external metadata from DBpedia** to enhance the factual depth of the graph (authors' biographies, birthplaces, and so forth).

1.3 Literature Review

Several studies have focused on building knowledge graphs for literary databases, often to power advanced book recommendation systems. For instance, **?** proposes a semantic graph for literary works, while **?** discusses the use of language models for sentiment analysis in book reviews. Our approach aligns with these efforts, leveraging recent advancements in LLMs and the rich structured data from DBpedia.

2 Methodology (Team Work)

2.1 Team Organization

- **Adam:** Responsible for data preparation and cleaning, as well as graph schema design. This includes writing `preprocess_data.py` to remove duplicates, handle missing fields, and generate consistent CSVs ready for ingestion.
- **Rayan:** In charge of data enrichment via LLMs. Implemented `LLMGraphEnrichment` (in `LLM_integration.py`) and `LLMComparaison` (in `LLM_comparaison.py`), which interact with `OpenRouter` POST requests to communicate with different free models in a centralized way and using a single API Key. This was done to generate descriptions, genres, themes, and similarity links for books, as well as applying comparaison methods and metrics between four different models. Prompt Engineering was also studied in order to optimize LLM responses.
- **Sami:** Focused on integrating cleaned data to the Knowledge Graph using `Cypher` queries, as well as metadata from DBpedia, such as author biographies or adaptation links. Designed precise `SPARQL` queries to finalize the enriched graph. The core logic resides in `DBPedia_integration.py`.

2.2 Collaborative Methods

To coordinate work and ensure code traceability:

- **GitHub** was used for code hosting, issue management, and version control.
- **Regular Meetings** (every 2 days) were scheduled to synchronize progress and address technical hurdles.
- **Asynchronous Communication** via Whatsapp for smaller updates and resource sharing.

2.3 GitHub Link

The complete source code, including test notebooks and additional documentation, is available in a public GitHub repository:

`https://github.com/Samsam19191/GoodReadsKG_LLM`

3 Materials and Methods

3.1 Technologies Used

- **Python**: Our primary programming language for data cleaning, graph creation, and enrichment tasks.
- **Neo4j**: A graph-oriented database used to store nodes (`Book`, `Author`, `User`) and relationships.
- **OpenRouter API**: Accessed through POST HTTP requests using the `requests` library to leverage in a centralized way various free language models.
- **SPARQL** (via `SPARQLWrapper`): Employed to query DBpedia for additional metadata.

Neo4j was chosen for its performance and flexibility in managing highly connected data, while the OpenRouter API offers many free language models to use wihtout having to manage multiple API Keys. DBpedia provides a robust RDF knowledge base containing rich information about authors, works, places, and more.

3.2 Detailed Methodology

3.2.1 Data Preparation and Cleaning

Data Sources We received Goodreads data in CSV format:

- **Books CSV**: Contains *Title*, *Authors*, *Publisher*, *Language*, *Rating*, etc.
- **Ratings CSV**: Contains entries mapping a `User` ID to a `Book` ID and a textual *Rating*.

Cleaning Pipeline Using `pandas`, we perform:

1. **Load with `pandas`**: Read CSV files into DataFrames.
2. **Check Columns**: Ensure required fields exist (*Id*, *Name*, *Rating*, etc.).
3. **Handle Missing Values**: Replace missing data for `Author`, `Publisher`, etc., with placeholders or inferred values.

4. **Filter and Map Ratings**: Convert textual user ratings (e.g., *"did not like it"*, *"it was ok"*) to numeric scales.

5. **Export Clean CSVs**: Write to `cleaned_books.csv` and `cleaned_ratings.csv`.

Partial Sentiment Analysis (Textual Ratings to Numeric) In our dataset, user ratings were provided as categorical textual responses such as *"did not like it"*, *"it was ok"*, *"liked it"*, *"really liked it"*, and *"it was amazing"*. Due to the absence of more detailed textual reviews or free-form comments, we employed a straightforward mapping approach to convert these categorical ratings into numerical values:

```
rating_map = {
   "did not like it": 1,
   "it was ok": 2,
   "liked it": 3,
   "really liked it": 4,
   "it was amazing": 5,
}
self.df["NumericalRating"] = self.df["Rating"].map(rating_map)
```

While this is not a traditional sentiment analysis pipeline that infers sentiment from free-text reviews, it effectively captures user sentiments through predefined categorical responses. This approach is advantageous because it directly utilizes the user-provided ratings without the need for complex natural language processing. The absence of more granular textual reviews in the dataset made this approach both practical and sufficient for quantifying user opinions.

3.2.2 Graph Construction in Neo4j

Schema Design The `GraphCreator` class (in `define_kg.py`) initializes a Neo4j driver and defines the following schema:

- `Book` nodes: storing properties like `id`, `name`, `rating`, `language`, `description`, etc.
- `Author` nodes: storing the author's `name`.
- `User` nodes: storing the user's `id`.

Relationships:

- `(b:Book)-[:WRITTEN_BY]->(a:Author)`

- `(b:Book)-[:REVIEWED_BY rating, num_rating]->(u:User)`

Implementation Details

- **generate_book_graph**: Reads the cleaned CSV and merges `Book` and `Author` nodes while linking them with `WRITTEN_BY`.
- **add_ratings_to_graph**: Iterates over rating entries to `MERGE User` nodes and create the `REVIEWED_BY` relationship with the user's numeric rating.

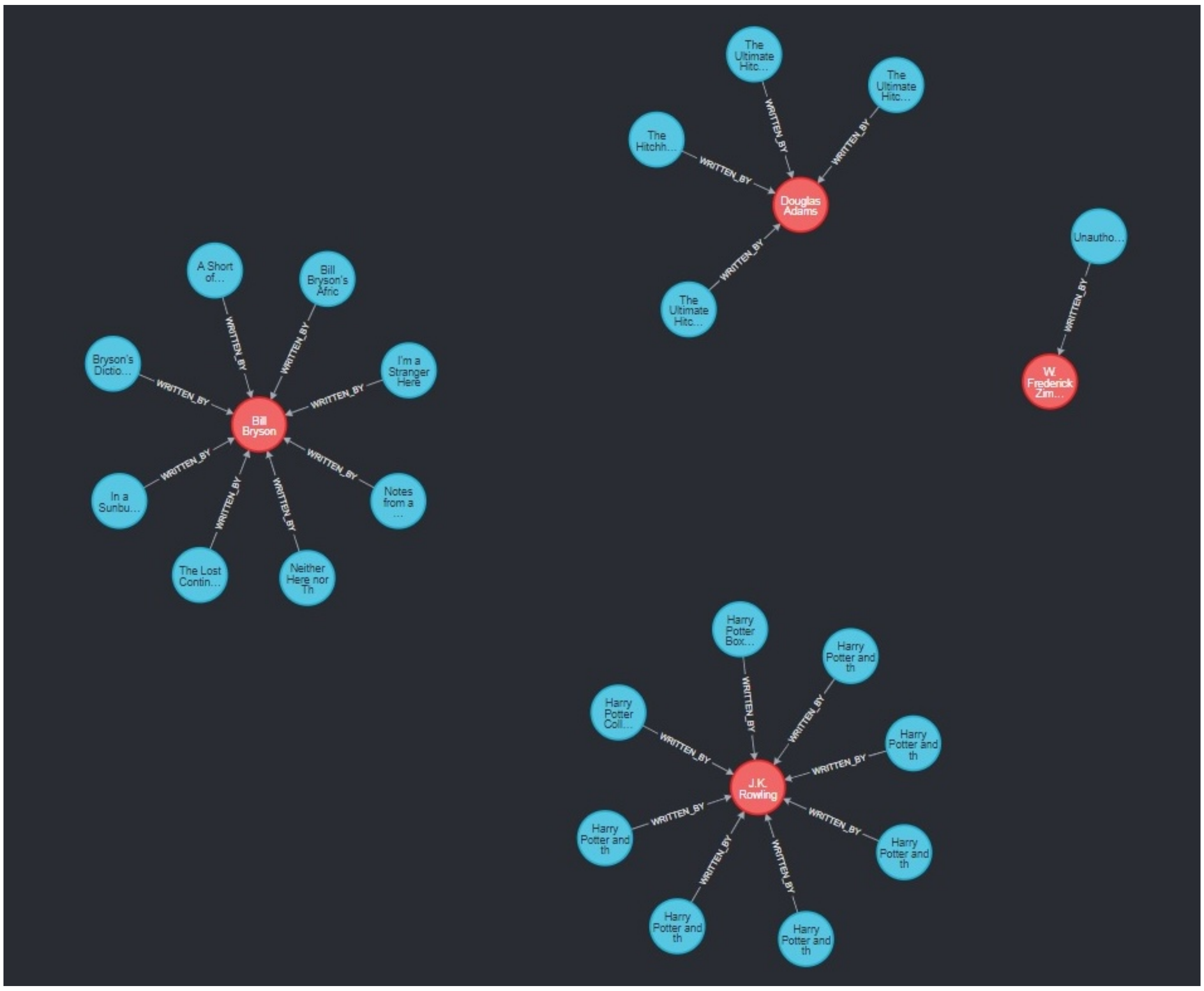

Figure 1: A sub-graph of some authors and their books.

3.2.3 LLM-based Enrichment

LLM Comparaison The `LLMComparaison` class in `LLM_comparaison.py` uses the Open-Router API to query five different Language Models. We will compare the LLMs on the attribute creation based on description analysis. The results will be analyzed in this paragraph using two metrics : runtime and accuracy. The replies from the LLMs were get by sending the following prompt:

You are a Data Scraper and a Data Engineer for a famous online bookstore. You will be tasked with analyzing books data in order to provide attributes or relationships.
Be sure to never make up any information. When you are not sure with what to reply, do not try to make something up.
Here is your task :

You will analyze the description of a book, given its title and author. You have to extract the genres, the themes, and the target audiences of the book. Each has to be a list (in brackets : []), even if it is a single genre or a single theme or a single audiance. Provide the reply in the following format:
Genre: [genres], Themes: [themes], Audience: [audiences].
If you cannot determine one or many of these attributes, put "unknown" in the brackets. Do NOT add anything to your reply apart from the attributes.

Here is a reply example for the book titled "Harry Potter and the Philosopher's Stone" by "J.K. Rowling" having the description "Harry Potter is an ordinary boy who lives in a cupboard under the stairs at his Aunt Petunia and Uncle Vernon's house, which he thinks is normal for someone like him who's parents have been killed in a 'car crash'. He is bullied by them and his fat, spoilt cousin Dudley, and lives a very unremarkable life with only the odd hiccup (like his hair growing back overnight!) to cause him much to think about. That is until an owl turns up with a letter addressed to Harry and all hell breaks loose! He is literally rescued by a world where nothing is as it seems and magic lessons are the order of the day. Read and find out how Harry discovers his true heritage at Hogwarts School of Wizardry and Witchcraft, the reason behind his parents mysterious death, who is out to kill him, and how he uncovers the most amazing secret of all time, the fabled Philosopher's Stone! All this and muggles too. Now, what are they?"
Genre: [Fantasy], Themes: [Magic, Friendship, Adventure], Audience: [Children, Young Adults].

Here is the data you need to work on :
Title : "{title}", Author : "{author}", Description : "{description}"

The attributes created (`Genres`, `Themes`, and `Audience`) by the LLMs for two different

Books are summarized right below :

- The table below contains the results for the book "*Lessons Learned (Great Chefs, #2)*" by **Nora Roberts** having the description "*LESSONS LEARNED... Coordinating the publicity tour for Italy's most famous and most adorable chef was just the kind of assignment Juliet relished. Carlo Franconi could gather a crowd just by smiling, and watching him prepare a meal was like witnessing a lesson in passionate lovemaking. By the time the tour was over, Juliet planned to have Carlo packaged as the world's sexiest chef. Women everywhere would fantasize about him preparing an intimate meal for two. But Juliet hadn't counted on being part of the dinner plans. Candlelight, pasta and romance... Carlo distracted her with his charms, setting his romantic recipes simmering in her heart.*":

Model	Genres	Themes	Audience	runtime
Google's LearnLM 1.5 Pro Experimental	Romance	Cooking, Travel, Love	Adults	2.123 s
Microsoft's Phi 3 Medium 128k	Romance, Culinary	Cooking, Love, Passion	Adults	1.930 s
Meta's Llama 3.2 3B	Romance, Cooking	Love, Romance, Food	Adults	1.427 s
Google's Gemini 2.0 Flash Experimental	Romance	Love, Cooking	Adults	1.985 s
HuggingFace's Zephyr 7B Beta	Romance	Cooking	Adult	5.073 s

Table 1: Table of LLM responses for the first book

We can observe a terrible performance from `Zephyr` model in terms of runtime. We can already eliminate it. But apart from this one, all the models seem to be quick enough, with a slight better performance from `Llama 3.2`. In terms of accuracy, we can see that `Phi 3` and `Llama 3.2` are wrong when they specify "Culinary" and "Cooking" in Genres. The book indeed has nothing to do with a culinary book, according to readers on GoodRead's reviews section, apart from the fact that the main character is a Cook ! We can also remark that `LearnLM 1.5` has the great ability here to not repeat itself in terms of Themes, and even to specify Travel which is a good guess for this book. `Gemini 2.0` also achieved to not repeat two themes, by having only Love and not Romance or Passion associated with it.

We could conclude here that `Gemini 2.0` is a good bridge between the speed of `Llama 3.2` and the accuracy of `LearnLM 1.5`.

- The table below contains the results for the book "*Walking by Faith: Lessons Learned in the Dark*" by **Jennifer Rothschild** having the description "*At the age of fifteen, Jennifer Rothschild confronted two unshakable realities: Blindness is inevitable ... and God is enough. Now this popular author, speaker, and recording artist offers poignant lessons that illuminate a path to freedom and fulfillment. With warmth, humor, and insight, Jennifer shares the guiding principles she walks by – and shows you how to walk forward by faith into God's marvelous light.*":

Model	Genres	Themes	Audience	runtime
Google's LearnLM 1.5 Pro Experimental	Christian Living, Inspirational	Faith, Blindness, Adversity, Acceptance	Adults	1.749 s
Microsoft's Phi 3 Medium 128k	Faith, Inspiration, Management, Social Responsibility	Faith, Overcoming Adversity, Personal Growth, Ethical Leadership, Corporate Social Responsibility, Social Change	Adults, Inspirational Readers, Professionals, Business Owners, Managers	15.930 s
Meta's Llama 3.2 3B	unknown	unknown	unknown	1.319 s
Google's Gemini 2.0 Flash Experimental	Christian	Faith, Spirituality, Personal Growth	Adults	2.159 s

Table 2: Table of LLM responses for the second book

We can observe a terrible performance from `Phi 3` : not only it took 15s to execute, but he also messed everything up by querying from the Internet **another** description for this book, but the description included inspirations of the Author that were about leadership and corporations for some reason...
However, in another try, it returned way more accurate results :

Genres	Themes	Audience	runtime
Christian, Non-fiction	Faith, Disability, Personal Growth	Adults	1.721

The unstability `Phi 3` is still a reason to exclude it.
`Llama 3.2` being this time again the fastest, it did not return any information from the description, and this two consecutive times ! This can be proof that it listens carefully to the prompt as it was told not to make anything up. It probably realized that this book was not in its training data therefore did not give anything back.
We therefore tried another time by taking out from the prompt the parts where we tell to not make anything up, and got excellent results : The execution time is

Genres	Themes	Audience	runtime
Faith, Spirituality	Freedom, Fulfillment, Hope	Adults, Christians	1.297

impressive, and the accuracy is also ! A problem that all three remaining LLMs though, is the presence of "Christian" in one of the attributes. **J. Rothschild** is indeed a Chritian writer, but this information is given nowhere in the description. This can be excused for `Gemini 2.0` `LearnLM 1.5` as they might still have access to this information, but it is a problem for `Llama 3.2` that supposedly only deduced the attributes from the description. Knowing only the description, which contains references to God, Light and lots of Monotheistic concepts, we could deduce that the book is for a Muslim or Jew audience, but `Llama 3.2` still returns Chirstians. Appart from this minor issue, it returns excellent accuracy for Genres and Themes. The two other models also perform very well in terms of accuracy, maybe with more precise Themes and a better runtime from `LearnLM 1.5`.

To conclude, we might be tempted to use either `Llama 3.2`, `LearnLM 1.5` or `Gemini 2.0` for our final Enrichment, with the first being quicker but the two last being more accurate. We can eliminate `Gemini 2.0` because the free-queries quota of this last model is very low, which is a major issue regarding the amount of data we want to Enrich.

To choose between the two remaining models, we will compare them in the hard field of description generation using only the title, the author, the publisher and the publication year. The models might hallucinate a lot for this task, and it is why we need to the right choice here. We will compare them on two books that already have a description to use it as a metric.
The prompt used is the following :

You are a Data Scraper and a Data Engineer for a famous online bookstore. You will be tasked with analyzing books data in order to provide attributes or relationships.
Here is your task :

Provide, from whichever source you have access to (Internet, training data, etc.), a description for the book titled "{title}" by "{author}". If you do not recognize the book or cannot provide a description, respond with "none". Do not make a description up without knowing the book. Do not give any other information than the description or the string "none".

We have the following results :

- The first book is "*Computing Calamities: Lessons Learned from Products, Projects, & Companies That Failed*" by **Robert L. Glass** :

Model	Description	runtime
Google's LearnLM 1.5 Pro Experimental	Computing Calamities examines a collection of failed software products, projects, and companies, dissecting the reasons behind their demise. Through detailed case studies, Robert L. Glass identifies common patterns of mistakes and misjudgments in software development, offering valuable lessons for practitioners and managers to avoid similar pitfalls. The book explores a wide range of factors contributing to failure, including technical challenges, managerial incompetence, market miscalculations, and organizational dysfunction. Glass emphasizes the importance of learning from past failures to improve future software endeavors, providing insights into risk management, project planning, and effective leadership.	4.197 s
Meta's Llama 3.2 3B	Computing Calamities: Lessons Learned from Products, Projects, & Companies That Failed" by Robert L. Glass is a book that focuses on the lessons learned from failed products, projects, and companies in the computing industry, highlighting common pitfalls and best practices for avoiding similar failures in the future.	1.866 s

Table 3: Table of LLM responses for the first book

For accuracy comparaison purposes, here is the actual description from GoodReads for this book:

> *Many great advances in technology have resulted from risky experimentation, but it's critical to remember and study the spectacular failures that also resulted from some of those risks. Failures can be mundane, like the typical complaints of software projects that are behind schedule and over budget, while others can be much more extravagant. In Computing Calamities, Robert L. Glass has collected war stories from around the industry. Laugh at these mistakes, and learn from them. Someone else's failure could be the foundation of your success.*

We can observe that both models give a very satisfying description, probably because of the very explicit title ? We might prefer the `LearnLM 1.5`'s version in terms of clarity of speach and preciseness, but `Llama 3.2` gives its result more than two times quicker ! We will try another book to confirm our results so far.

- The second book is "*Rival Attractions*" by **Penny Jordan**:

Model	Description	runtime
Google's LearnLM 1.5 Pro Experimental	Rival Attractions tells the story of a woman named Gaby who finds herself torn between two men	3.688 s
Meta's Llama 3.2 3B	I was unable to find a description for the book "Rival Attractions" by "Penny Jordan".	1.866 s

Table 4: Table of LLM responses for the first book

For comparaison purpose, here is the original description of this book :

> *'You'd find life much less fraught if you learned to trust people a little, Charlotte. You're always so ready to believe the worst of others ...' But Charlotte Spencer was scared to let herself follow her instincts where Oliver Tennant was concerned. How could she respond to him as an attractive man when he was also a business rival who might be playing a deeper game? In any case, what had a country bumpkin like herself to offer a sophisticated man about town? Better by far to put all thoughts of love aside ...*

We see here that it was indeed because of the first book's title's explicity, because the results this time are terrible. Indeed, `LearnLM 1.5` totally makes up a story and a character that have nothing to do with the actual book. `Llama 3.2` has at least the merit of returning that it does not know, even though it is still not done the right way because it was explicitly dicted multiple times in the prompt to return "none" and nothing else.

To conclude, we will choose Meta's `Llama 3.2` because it performed better overall.

Description Generation After the not so convincing results of the description creation, we decided to assign this specific enhancment to DBPedia rather than the LLMs.

Attribute Extraction The `LLMGraphEnrichment` class in `LLM_integration.py` uses the OpenRouter API to query Meta's `Llama 3.2` LLM. If a description is available, the script prompts the LLM to extract *genre*, *themes*, or *target audience*. The returned text is parsed, and the results are assigned as node properties in Neo4j.

Similarity Relationships To create `SIMILAR_TO` edges:

1. The script queries the LLM with the known descriptions: *"Based on this description, suggest up to 3 similar books."*
2. For each returned title found in the graph, a `(b1)-[:SIMILAR_TO]->(b2)` relationship is `MERGE`d if it does not already exist.

Though somewhat subjective, it enriches the graph with cross-book references for recommendation purposes.

3.2.4 DBpedia Integration

SPARQL Queries In `DBPedia_integration.py`, we use `DBpediaConnector` (from `DBPedia_manager.py`) with queries like:

```
PREFIX dbo: <http://dbpedia.org/ontology/>
PREFIX foaf: <http://xmlns.com/foaf/0.1/>

SELECT ?birthPlace WHERE {
   ?author a dbo:Person ;
           foaf:name "Author Name"@en ;
           dbo:birthPlace ?birthPlace .
}
```

The results are converted into JSON, then integrated into the graph.

Adding Factual Properties and Links

- **(a:Author).biography**: If found in DBpedia, inserted as the property `biography`.
- **(b:Book)-[:HAS_GENRE]->(g:Genre)**: Books can link to multiple genres.

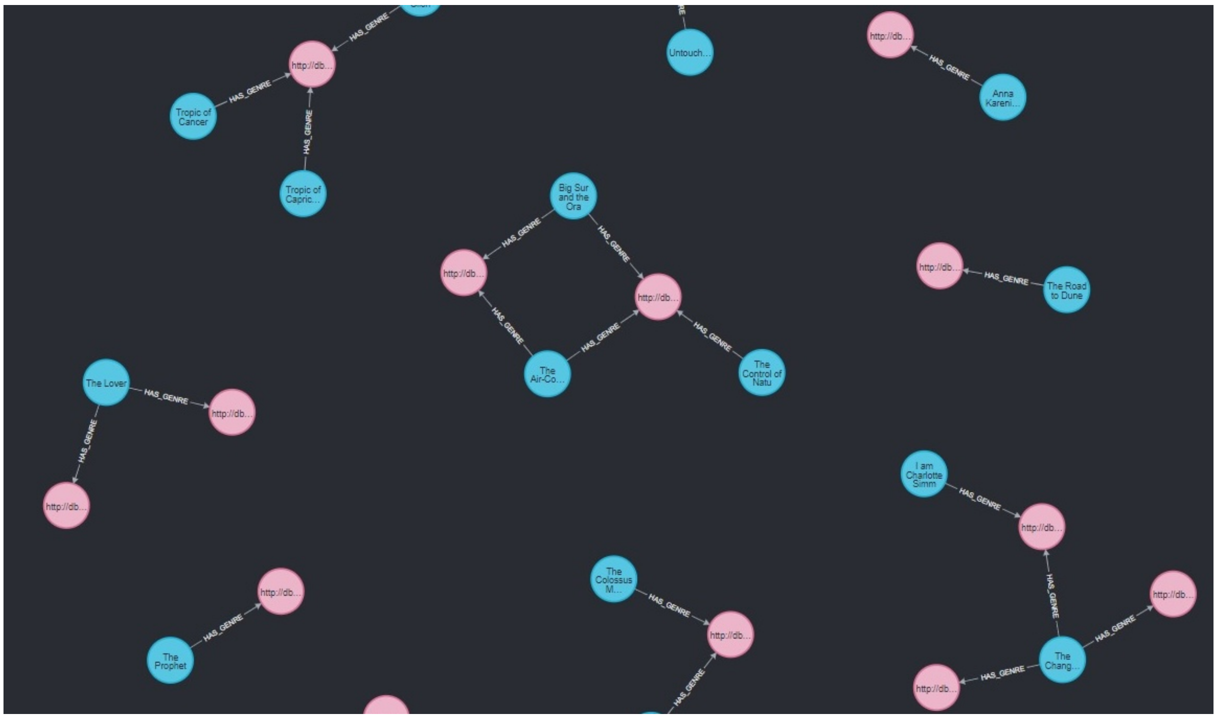

Figure 2: Genres-Books sub-graph

- **(b:Book)-[:HAS_ADAPTATION]->(a:Adaptation)**: E.g., film or TV versions.
- **(b1:Book)-[:SUBSEQUENT_EDITION]->(b2:Book)**: Tracks the evolution of certain works.
- **(b:Book).description**: If found in DBPedia, inserted as the property `description`
- **(b:Book)-[:HAS_SUBJECT]->(s:Subject)**: Subjects here are the equivalent of the themes queried from the LLMs. When found in DBpedia, the relationship is created.

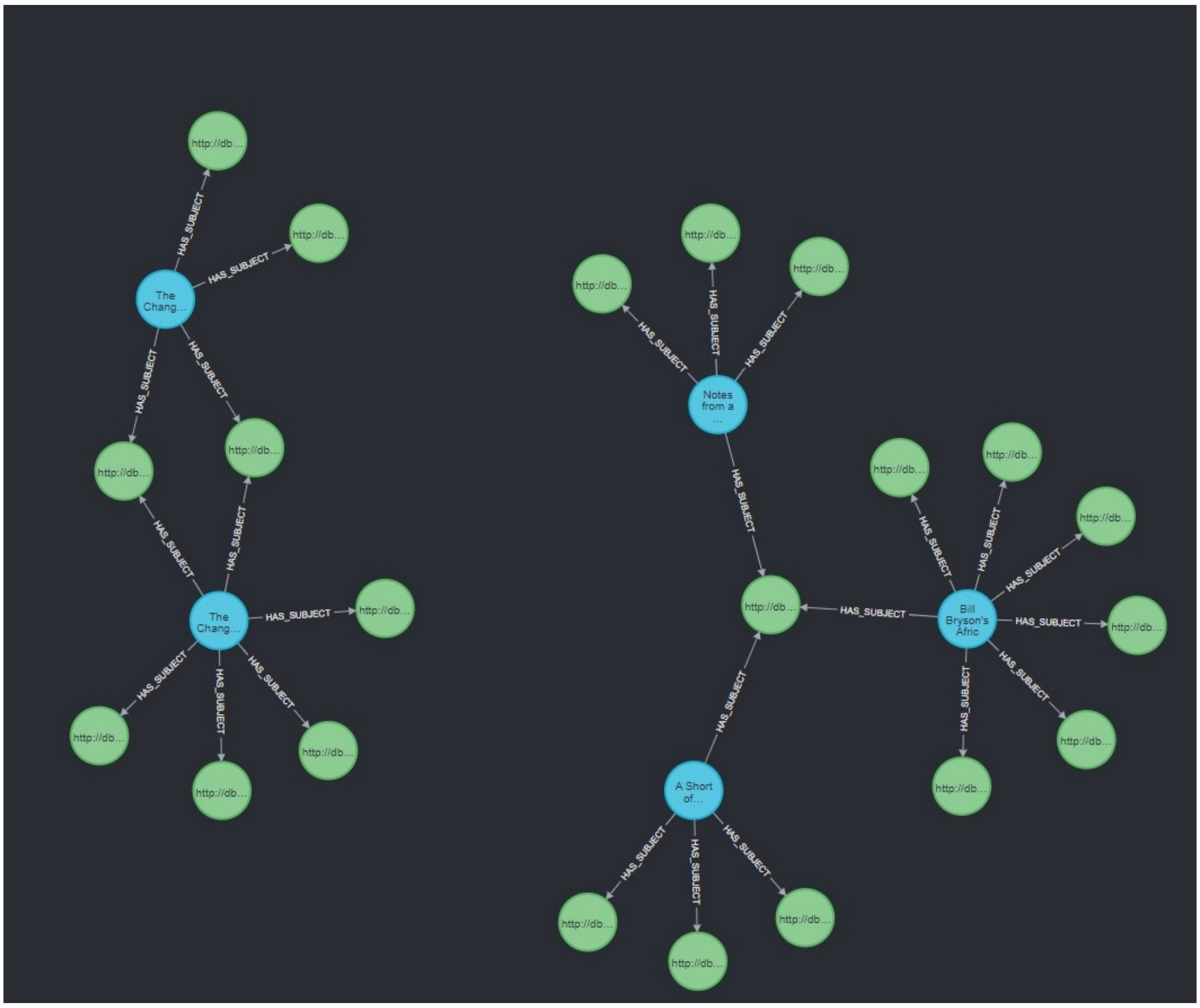

Figure 3: Subjects-Books sub-graph

3.2.5 Implementation Overview

In `main.py`:

1. **Data Preparation**: Processes CSVs into cleaned outputs.
2. **Graph Creation**: Inserts `Book`, `Author`, and `User` nodes, plus fundamental relationships.
3. **LLM Enrichment**: Adds descriptions, attributes, and `SIMILAR_TO` edges.
4. **DBpedia Enrichment**: Fetches external factual data via SPARQL and merges them into Neo4j.

Connections to Neo4j are wrapped in `try-finally` blocks to ensure the driver session is closed properly.

4 Results

4.1 Added Descriptions and Attributes

- **DBPedia-fetched Descriptions and Relationships**: Approximately 80% of sampled books lacked a prior description. We added to most of them (50 to 60%) a description attribute.
- **LLM-inferred Genres, Themes, Audience and similarities**: Extracted from both the LLM and DBpedia. We found that ≈30% of books ended up with multiple genres assigned, which can greatly aid in recommendation queries.

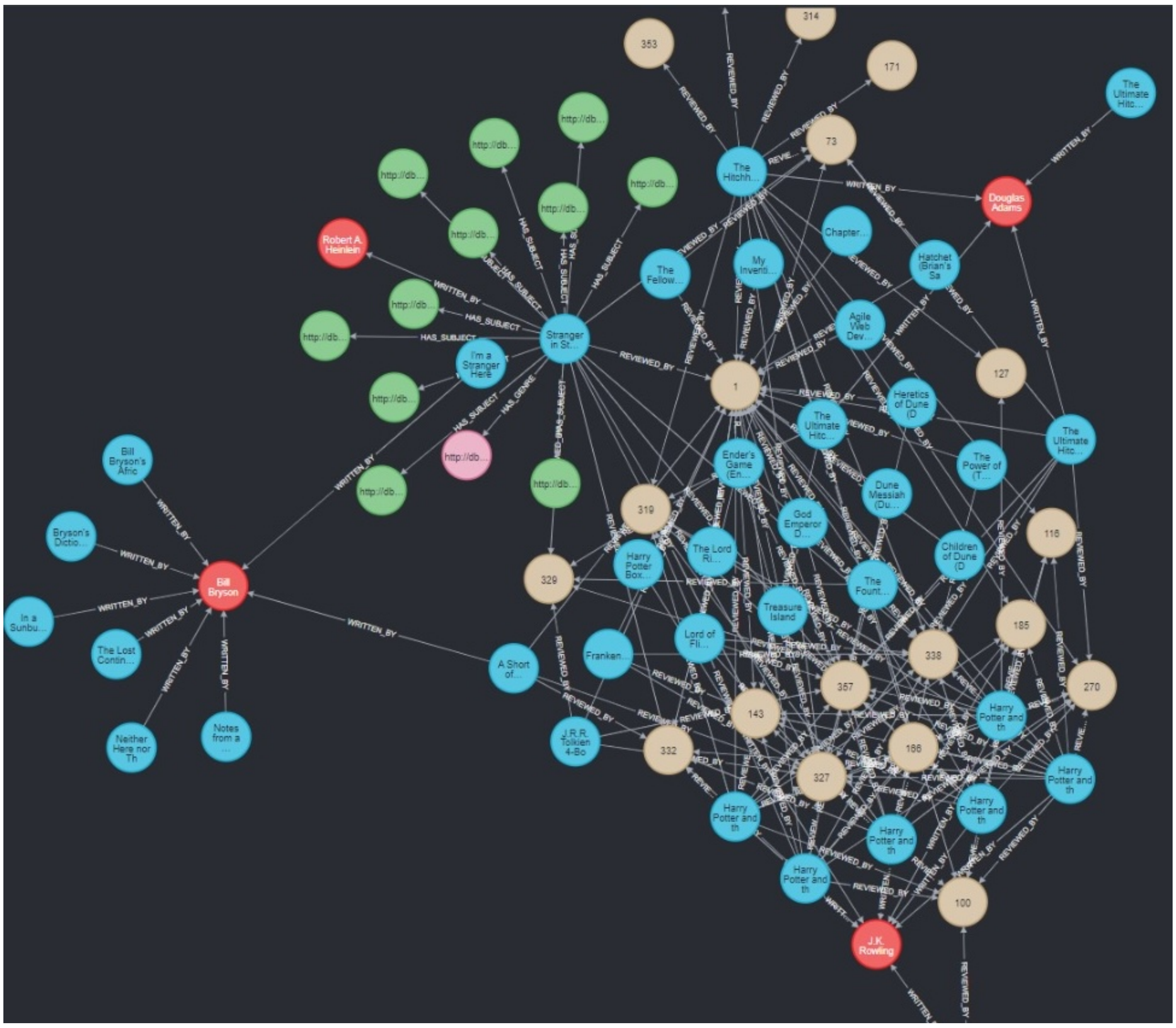

Figure 4: Semi-global KG overview

4.2 DBpedia Facts and Adaptations

- **Biographies, Birthplaces**: Almost all of them, around 70-80% of authors had a matching `foaf:name` entry in DBpedia with a retrievable `abstract` or `birthPlace`. An additional 10% had partial matches due to naming variations or missing language tags.
- **Description**: Lots of books had a matching `foaf:name` entry, and hence a description attribute could be created by retrieving the associated `abstract`.
- **Edition Links**: For classic or well-known literary works, DBpedia provided `SUBSEQUENT_EDITION` or `PRECEDING_EDITION` relationships, allowing us to trace the evolution of these works.
- **Adaptations**: Over 2k books were linked to `Adaptation` nodes representing film or television versions.

4.3 Example Queries and Potential Use Cases

4.3.1 Neo4j Example

```
-- Cypher example: find books similar to a certain classic
MATCH (b:Book {name: "Pride and Prejudice"})-[:SIMILAR_TO]->(sim:Book)
RETURN sim.name AS SimilarTitles;
```

This query returns all books that our LLM-based process flagged as similar to Pride and Prejudice*.*

```
-- Cypher example: find books of a specific Genre
MATCH (book:Book)-[:HAS_GENRE]->(genre:Genre {name: "Romance"})
RETURN book.title AS BookTitle, book.author AS Author;
```

This query returns all books having the genre : Romance

5 Discussion and Conclusion

5.1 Analysis of Results

Our integrated knowledge graph successfully merges Goodreads user-generated data (ratings, basic book metadata) with both **LLM-derived** textual enrichment and **DBpedia-**

based factual data. This synergy leads to:

- **Wider coverage of book properties**, enabling deeper semantic queries.
- **Enriched recommendation potential**, bolstered by textual `SIMILAR_TO` relationships.
- **Improved factual accuracy**, especially for author details and edition relationships.

Additionally, our approach to mapping categorical textual ratings to numerical values effectively captures user sentiments without the need for complex natural language processing. This method leverages the structured nature of Goodreads' rating system, providing a straightforward and reliable measure of user opinions directly from their responses.

5.2 Limitations and Future Directions

- **LLM-Related Costs and Latency**: Large-scale usage of calls can be expensive and time-consuming. Implementing caching mechanisms or focusing enrichment efforts on popular books first could mitigate these issues.
- **DBpedia Coverage**: Some authors or less mainstream books are missing or labeled differently in DBpedia. Integrating additional sources like Wikidata could enhance coverage.
- **Subjective Similarities**: The `SIMILAR_TO` relationships depend on the LLM's textual reasoning, which can introduce subjectivity. Exploring hybrid recommendation models that combine collaborative filtering with content-based methods might yield more accurate and reliable recommendations.
- **Scalability**: For processing millions of entries, bulk import methods and parallel processing strategies are essential to maintain practical processing times.
- **Full Sentiment Analysis**: Although our mapping of categorical ratings effectively captures user sentiment, incorporating a dedicated sentiment analysis pipeline for any available free-text reviews could provide more nuanced insights. However, the absence of extensive free-text reviews in the dataset limited our ability to implement such a feature in the current project.

6 Appendices

6.0.1 Scripts and Configuration Examples

- `preprocess_data.py`: Loading and cleaning book/rating data.
- `define_kg.py`: Creating nodes and relationships in Neo4j.
- `LLM_integration.py`: Enrichment with descriptions, attributes, and similarities using the OpenRouter API.
- `LLM_comparaison.py`: Comparaison of models using the OpenRouter API.
- `DBPedia_integration.py`: SPARQL queries for additional metadata (biographies, adaptations, etc.).
- `neo4j_manager.py`: Manages Neo4j connections and Cypher queries.
- `DBPedia_manager.py`: Handles SPARQL queries to DBpedia.
- `main.py`: Orchestrates data preparation, graph creation, LLM enrichment, and DBpedia enrichment.

6.0.2 Example `.env` Configuration

```
NEO4J_URI=bolt://localhost:7687
NEO4J_USERNAME=neo4j
NEO4J_PASSWORD=pass123
OPENROUTER_API_KEY=sk-....
```

Conclusion

In conclusion, our integrated pipeline for constructing a Goodreads knowledge graph effectively leverages both LLM-based textual enhancements and DBpedia-based factual data to fill gaps in coverage and metadata richness. Through evolutions in terms of sentiment analysis and subjective similarity, the knowledge graph can become an even more powerful tool for literary data exploration and recommendation.