Research Paper Graph Creation with DIFY

Md Assaduzzaman Samrat
Matriculation No: 428968
RPTU Kaiserslautern, Department of Computer Science

***Note:** This report contains a project documentation and reflection on the portfolio task submitted for the lecture Engineering with Generative AI in WiSe 2024-25. This report is an original work and will be scrutinised for plagiarism and potential LLM use.*

1 Portfolio documentation

The main objective of this task is to utilize DIFY, Large Language Models (LLMs), and API tools to extract metadata from the provided research paper and perform necessary enhancements of extracted data and finally generate a knowledge graph. The workflow includes extraction of data and summarizing text using LLMs, restructuring and enhancing extracted data with API tools, and organizing it into a graph format. The project involves:

- Selecting an appropriate unpaid LLM compatible with DIFY.
- Choosing a free API tool for metadata extraction.
- Enhancement of retrieved data from the API.
- Implementing a graph knowledge base engine.
- Designing prompts to enhance model performance.
- Structuring an efficient project workflow.

1.1 Research Phase

In this phase, three fundamental components are essential for integration with DIFY:

1.1.1 Unpaid LLM with Large Context Length

An open-source, unpaid LLM with a large context window is needed to process entire research papers effectively. Therefore, "mistralai/Mistral-7B-Instruct-v0.3" [1] LLM model is perfectly compatible with the given requirements. The selected model was considered based on processing speed, accuracy, and capability to extract relevant details.

1.1.2 Free API Tool for Metadata Extraction

Semantic Scholar API [2] tool was selected to extract metadata from research papers, including the title, authors, abstract, venue, year, DOI, and references. The API selection criteria included:

- Free and easy access.
- Standard academic formats.
- Appropriate data fetch assurance.
- Efficient performance in terms of accuracy and data format.

1.1.3 Graph Knowledge Base Engine

A graph database engine was selected to store extracted information. Neo4j [3]was chosen for its:

- Native graph storage and query capabilities.
- Compatibility with DIFY.
- Scalability for large datasets.
- excellent visualization and analyzing capacity.

1.2 Design Phase

This phase involved designing the schema for the knowledge base, including nodes, properties, and relationships.

1.2.1 Schema Design

According to the given requirement, the knowledge graph must consist of at least four distinct nodes. Therefore we have designed 5 distinct nodes:

- **Paper**: Represents a research paper and includes properties such as title, abstract, DOI, paperId, and publication year.
- **Author**: Contains the full name of the author.
- **DOI**:DOI (Digital Object Identifier) is a unique alphanumeric string assigned to identify a specific piece of content, such as a journal article, research paper, report, dataset, or other academic resources.
- **FieldsOfStudy**: Describes the field of the research paper.
- **Venue**: Represents the place or platform where the paper was published.

One node, **Paper**, includes at least five properties:

- Title
- DOI
- Abstract
- Publication Year
- PaperId

1.2.2 Relationships in the Knowledge Graph

The designed schema establishes meaningful relationships between these nodes to capture essential research metadata. The relationships are structured as follows:

- **WROTE**: Connects an **Author** to a **Paper**, indicating authorship

$$\text{Author} \xrightarrow{\text{WROTE}} \text{Paper}$$

- **PUBLISHED_IN**: Links a **Paper** to a **Venue**, showing where the research was published

$$\text{Paper} \xrightarrow{\text{PUBLISHED\_IN}} \text{Venue}$$

- **PUBLISHED_BY**: Links one or more than one **Paper** to a **Venue**, showing how many and what papers published by this venue

$$\text{Venue} \xrightarrow{\text{PUBLISHED\_BY}} \text{Paper}$$

- **HAS_DOI**: Associates a **Paper** with a **DOI**, ensuring a unique identifier for citation tracking

$$\text{Paper} \xrightarrow{\text{HAS\_DOI}} \text{DOI}$$

- **RELATES_TO**: Connects a **paper** to one or more **FieldsOfStudy**, categorizing the research into relevant disciplines

$$\text{Paper} \xrightarrow{\text{RELATES\_TO}} \text{FieldsOfStudy}$$

.

1.2.3 LLM and API Integration

Decisions were made regarding which components would be extracted using the LLM and which via the API:

- LLM: Extracted summaries, key topics, sample metadata show, and the overview of generated graph.
- API: Retrieved metadata and references.

1.2.4 Graph Knowledge Base Engine

- Open-source graph database for structuring extracted information into a knowledge graph. Neo4j [3] graph database was selected for this part.

1.3 Implementation Phase

The implementation of the knowledge extraction and graph generation process was carried out using **Dify**, where different components were structured into two parallel branches for processing research papers and their references.

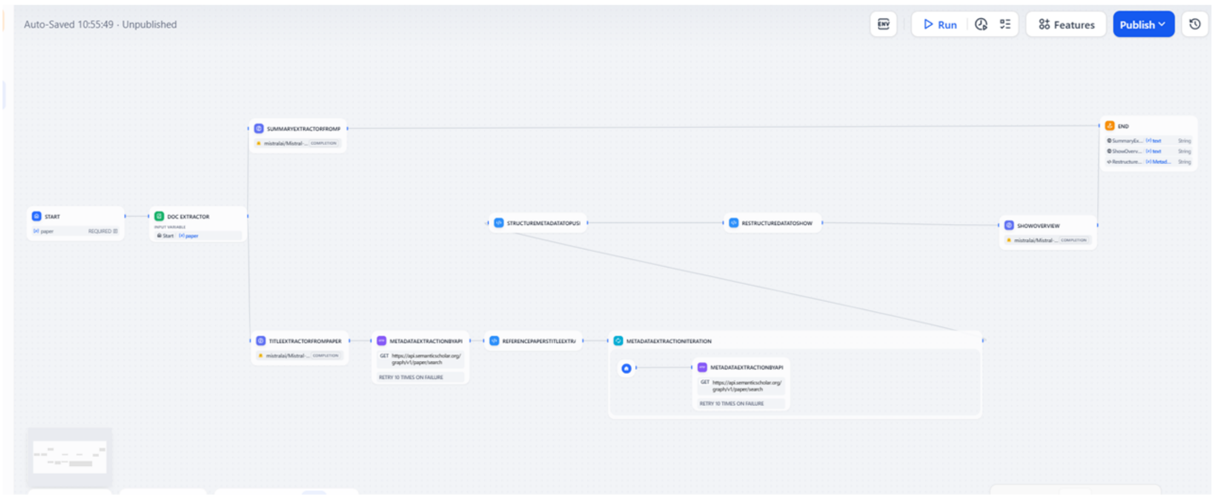

Figure 1: Workflow of the Implementation Process

1.3.1 Branch 1: Summary Extraction

The first branch focuses on extracting the summary of a research paper using the following sequence of components:

1. **Start**: Initiates the process by taking a paper as input.
2. **DocExtractor**: Extracts structured information from documents and the content of the research paper.
3. **LLM for Summary Extraction**: The label used for this component is SummaryExtractorFromPaper. I used mistralai/Mistral-7B-Instruct-v0.3 model from huggingface [4] for this step. It processes the extracted content and generates a concise 1-page summary. The following LLM prompt is used to extract the summary:

   mistralai/Mistral-7B-Instruct-v0.3 [1] `You are an intelligent assistant tasked with extracting important information from research papers. Please analyze the following paper text and extract the required information. Summary should be 1 page.`
   Required Outputs:
   Summary: Provide a concise summary of the paper.
   Once you have extracted this information, return it in the following format: `summary: context`
4. **End**: Shows the summary of the paper generated by the LLM to the user.

1.3.2 Branch 2: Title-Based Metadata Extraction and Knowledge Graph Generation

The second branch is responsible for extracting titles, retrieving metadata, and structuring the knowledge graph. The workflow is outlined below:

1. **Start**: Initiates the process by taking a paper as input.
2. **DocExtractor**: Extracts structured information from documents and the content of the research paper.
3. **LLM for Title Extraction**: The label used for this component is TitleExtractorFromPaper.I used mistralai/Mistral-7B-Instruct-v0.3 [1] model from huggingface [4] for this step. It processes the document and retrieves the research paper title. The following LLM prompt is used to extract the summary:

   > Given the following text from a research paper, extract the title of the paper. The title is typically located at the beginning of the document and may be followed by the authors' names and abstract. Return only the title, do not add any extra text with the title.
   > `Text form DocExtractor` Title

4. **HttpRequest API for Metadata Retrieval**: Fetches metadata from Semantic Scholar using the extracted title.
   a) **API Request Example**: To retrieve information for a specific paper, you can use the following query parameters in your request:

   ```
   GET https://api.semanticscholar.org/graph/v1/paper/search
       ?query=<title_of_the_paper>
       &limit=1
   ```

   This request will return data for the paper whose title most closely matches the query, limiting the response to only 1 paper.

   b) **Response Example**: A typical response from the API will look like this:

   ```
   {
       "data": [
           {
               "title": "Title of the Paper",
               "authors": [
                   {"name": "Author Name 1"},
                   {"name": "Author Name 2"}
               ],
               "abstract": "This is the abstract of the paper...",
               "venue": "Journal Name",
               "year": 2020,
               "references": [
                   {"title": "Reference Title 1"},
                   {"title": "Reference Title 2"}
   ```

```
                ]
            }
        ]
    }
```

5. **Python Code Block for Title Aggregation**: Combines the extracted paper title with the reference paper titles. The name used for this component is ReferencePapersTitleExtraction.

    a) **Python Script**:

    ```
    import json

    def main(response_json: str) -> dict:
        try:
            # Parse the outer JSON
            inner_data = json.loads(response_json)

            # Extract title of given paper
            Title = inner_data["data"][0].get("title")

            # Extract references
            references = inner_data["data"][0].get("references", [])

            # Extract only titles (ignoring null values)
            paper_titles = [
                {"title": ref["title"]}
                for ref in references
                if ref.get("title")
            ]
            paper_titles.append({"title": Title})

            return {
                "result": paper_titles  # Returning an array of objects with titles only
            }

        except (json.JSONDecodeError, KeyError, IndexError) as e:
            return {
                "error": f"Failed␣to␣process␣JSON:␣{str(e)}"
            }

    # Example usage:
    # response_json = '...'  # Replace with actual JSON string
    # print(main(response_json))
    ```

    b) **Explanation**: This Python script processes a JSON response by extracting the title of the paper and the titles of the references. If the response contains valid titles, they are added to an array, which is then returned as the result. In case of any error in processing the JSON data, an error message is returned.

6. **Iteration Block for Metadata Retrieval**: Iterates through all titles extracted from the document and reference section. Inside this iteration:
   - **Start**: Begins the iteration.
   - **HttpRequest API**: Calls the Semantic Scholar API to fetch metadata for each listed title. It returns a list of JSON data as array of string.

7. **Python Code Block for Metadata Restructuring and Neo4j [3] API Integration**: Restructures the fetched metadata and prepares it for integration into the knowledge graph. The name used for this step is StructureMetadataToPushInNeo4jAPI.

   a) **Script Functionality**: The script processes a list of JSON strings, extracts the relevant details from each research paper, and returns a structured dictionary. It sends the extracted data to a Neo4j [3] database via Cypher queries for creating nodes and relationships.

   b) **Script Code**:

```
import json
import requests

def main(output_json_data: list) -> dict:
    """
    Processes a list of JSON strings, extracts research paper details,
    and returns a dictionary with structured paper data.
    """
    neo4j_payloads = []

    for output_item in output_json_data:
        try:
            parsed_data = json.loads(output_item)  # Ensure it's valid JSON
        except json.JSONDecodeError:
            print("Skipping␣invalid␣JSON␣entry:", output_item)
            continue

        papers = parsed_data.get("data", [])

        for paper in papers:
            paper_id = paper.get("paperId", "")
            title = paper.get("title", "")
            abstract = paper.get("abstract", "")
            fieldsOfStudy = paper.get("fieldsOfStudy", "")
            venue = paper.get("venue", "")
            year = paper.get("year", "")
            authors = [author.get("name", "Unknown") for author in paper.get("authors", []
            doi = paper.get("externalIds", {}).get("DOI", "")

            neo4j_payload = {
```

```
            "paperId": paper_id,
            "title": title,
            "abstract": abstract,
            "year": year,
            "venue": venue,
            "authors": authors,
            "doi": doi,
            "fieldsOfStudy":fieldsOfStudy
        }

        neo4j_payloads.append(neo4j_payload)

# Neo4j API endpoint
url = "https://4f05d51b.databases.neo4j.io/db/neo4j/query/v2"

# Headers
headers = {
    "Content-Type": "application/json",
    "Accept": "application/json",
    "Authorization": "Basic␣bmVvNGo6TU9uSjhCMXMxRjJkcXRiMlpjSTZUY29KaFZQSXprN0O0RXZzUk
}

# Request body with Cypher query for creating nodes and relationships
payload = {
    "statement": """
        UNWIND $papers AS paper
        MERGE (p:Paper {paperId: paper.paperId})
        SET p.title = paper.title,
            p.abstract = paper.abstract,
            p.year = paper.year,
            p.doi = paper.doi

        // Create Author nodes and WROTE relationship
        FOREACH (author IN paper.authors |
            MERGE (a:Author {name: author})
            MERGE (a)-[:WROTE]->(p)
        )

        // Create Venue node and PUBLISHED_IN relationship
        MERGE (v:Venue {name: paper.venue})
        MERGE (p)-[:PUBLISHED_IN]->(v)
        MERGE (v)-[:PUBLISHED_BY]->(p)

        // Create DOI node and HAS_DOI relationship
        MERGE (doi:DOI {id: paper.doi})
        MERGE (p)-[:HAS_DOI]->(doi)

        // Create FieldsOfStudy nodes and RELATES_TO relationship
        FOREACH (field IN paper.fieldsOfStudy |
            MERGE (f:FieldsOfStudy {name: field})
            MERGE (p)-[:RELATES_TO]->(f)
        )
    """,
    "parameters": {"papers": neo4j_payloads}
}
# Send the POST request
response = requests.post(url, headers=headers, json=payload)

# Print the response
if response.status_code == 200 or response.status_code == 201:
     return {"response": str(response.json())}
else:
     return {"response": str(response.text)}
```

c) **Explanation**: The script processes a list of JSON strings representing research papers. It extracts the relevant details like title, authors, abstract, venue, and more from each paper. It then sends this data to a Neo4j [3] database using a Cypher query. The query creates nodes for the paper, authors, venue, DOI, and fields of study, and establishes relationships between them. This allows the research papers to be stored in a graph database for further analysis and querying.

8. **Python Code Block for Metadata Processing**: Processes the metadata to create an overview for user presentation. The name used for this block is RestructureDataToShow.

   a) **Script Functionality**: The script takes a list of JSON strings, extracts relevant details from each paper (such as the paper's title, authors, abstract, venue, etc.), and structures this data into a dictionary format for further use.

   b) **Script Code**:

```
import json
import requests

def main(output_json_data: list) -> dict:
    """
    Processes a list of JSON strings, extracts research paper details,
    and returns a dictionary with structured paper data.
    """
    neo4j_payloads = []

    for output_item in output_json_data:
        try:
            parsed_data = json.loads(output_item)  # Ensure it's valid JSON
        except json.JSONDecodeError:
            print("Skipping␣invalid␣JSON␣entry:", output_item)
            continue

        papers = parsed_data.get("data", [])
        for paper in papers:
            paper_id = paper.get("paperId", "")
            title = paper.get("title", "")
            abstract = paper.get("abstract", "")
            fieldsOfStudy = paper.get("fieldsOfStudy", "")
            venue = paper.get("venue", "")
            year = paper.get("year", "")
            authors = [author.get("name", "Unknown") for author in paper.get("authors", []
            doi = paper.get("externalIds", {}).get("DOI", "")

            neo4j_payload = {
                "paperId": paper_id,
                "title": title,
                "abstract": abstract,
                "year": year,
                "venue": venue,
                "authors": authors,
                "doi": doi,
                "fieldsOfStudy":fieldsOfStudy
            }

            neo4j_payloads.append(neo4j_payload)

        return {
                "Metadata": str(neo4j_payloads)  # Returning an array of objects with titl
        }
```

c) **Explanation**: The script processes each JSON string in the input list, extracting key details like the title, authors, abstract, and other metadata from each research paper. It then returns the structured data as a dictionary. This structure can later be used to store the metadata in a database or for further analysis.

9. **LLM for Overall Overview**: Summarizes the processed metadata and describes graph structure. I used mistralai/Mistral-7B-Instruct-v0.3 [1] model from huggingface [4] for this step. The name of this step is ShowOverview. The following LLM prompt is used to get the overview:

   You are an expert in knowledge graph construction and academic data modeling. Your task is to describe structured Cypher queries and how they can populate a Neo4j [3] graph database with research paper metadata.

```
{
  "statement": """
    UNWIND $papers AS paper
    MERGE (p:Paper {paperId: paper.paperId})
    SET p.title = paper.title, p.abstract = paper.abstract, p.year = paper.year, p.

    // Create Author nodes and WROTE relationship
    FOREACH (author IN paper.authors |
      MERGE (a:Author {name: author})
```

```
          MERGE (a)-[:WROTE]->(p)
        )

        // Create Venue node and PUBLISHED_IN relationship
        MERGE (v:Venue {name: paper.venue})
        MERGE (p)-[:PUBLISHED_IN]->(v)
        MERGE (v)-[:PUBLISHED_BY]->(p)

        // Create DOI node and HAS_DOI relationship
        MERGE (doi:DOI {id: paper.doi})
        MERGE (p)-[:HAS_DOI]->(doi)

        // Create FieldsOfStudy nodes and RELATES_TO relationship
        FOREACH (field IN paper.fieldsOfStudy |
          MERGE (f:FieldsOfStudy {name: field})
          MERGE (p)-[:RELATES_TO]->(f)
        )
      """
    }
```

Based on the above Cypher query and metadata input string `Input metadata`, describe how many nodes and relationships the query will create in the knowledge graph. Also, include the `context` in your response.

Output format:

- **Nodes:** node 1 name, node 2 name, . . . , node n name
- **Relations:** relation 1, relation 2, . . . , relation n
- **Metadata:** provided input string

10. **End**: Prints the summary, generated knowledge graph, and sample metadata for users overview.

By structuring the workflow into these two branches, we efficiently extract key information, retrieve relevant metadata, and generate a well-structured knowledge graph in Neo4j [3]. This modular approach ensures scalability and adaptability for processing multiple research papers.

2 Reflection

Since for the first time I have used Dify for implementing this type of complex application, therefore I have faced a lot of problems and challenges. All the challenges and problems I have faced throughout the task are listed below:

2.1 Doc Extractor Related Problem

Initially, I was trying to process the uploaded file directly through the LLM, but I was not able to get my desired output, then I did some study on this, then I found a solution that if I use

Doc Extractor, then I can get the result. After finding this information, I tried with the Doc Extractor block and found my expected output.

2.2 API Selection Related Problem

At first, I selected the crossref API for extracting metadata and started implementation with this. Gradually, when I started in-depth implementation, then I realized that I was not getting my expected metadata through this API. As a result, I started finding a good API that can provide more precise data. Then, I found Semantic Scholar API, which is an unpaid API.

2.3 Code Block Related Problem

In the Dify code block, there is a binding that I must return a dictionary. But after getting data as a dictionary, I have faced a problem with processing this data. For instance, when I was returning a JSON string as a dictionary, I was not able to convert it to JSON in the HTTP Request block. Therefore, for data insertion in Neo4j I did an API call inside the code block.

2.4 Huggingface Account Related Problem

When I was implementing the workflow after completing a step, I had to test it. Because of frequent use of the LLM model, I exceeded my account's monthly limit. Therefore, I opened 4/5 more Hugging Face accounts and used those API keys to solve this problem.

2.5 Neo4j Graph and Dify Workflow Integration Related Problem

At the beginning I tried to integrate neo4j locally but I faced some errors. Then I posted the issue in the forum then got a better solution than that.

2.6 Answer of the Questions

1. What was the most interesting thing you learned while working on the portfolio? What aspects did you find interesting or surprising?

   Answer: One of the most interesting aspects of working on the portfolio was exploring the efficient use of components, particularly code blocks in DIFY. I found it especially valuable that within these code blocks, I could seamlessly call APIs, making HTTP requests optional in certain scenarios. This flexibility enhanced the overall workflow and facilitated data processing. Also, I found the component iteration very interesting. Because of its feature, I was able to do something like loop without any code. Additionally, I was surprised by the visualization capabilities of Neo4j. Seeing how simple Cypher queries could generate a well-structured, interactive graph of nodes and relationships was both surprising and insightful. This experience deepened my appreciation for graph databases and their ability to represent complex data in an intuitive and analyzable manner. It reinforced the power of graph-based approaches in knowledge representation and relationship mapping, further validating my decision to integrate Neo4j into the project.

2. Which part of the portfolio are you (most) proud of? Why? What were the challenges you faced, and how did you overcome them?

   Answer: I am particularly proud of the section where I restructured the data and wrote Cypher queries to generate the desired nodes and relationships in Neo4j. This aspect of the project was especially rewarding because it required careful data modeling and query optimization to ensure accuracy and efficiency. During the development process, I encountered several challenges and devised effective solutions to overcome them. Initially, I struggled to process uploaded files directly through the LLM, but after conducting research, I discovered that using the Doc Extractor provided the expected results. Similarly, while selecting an API for metadata extraction, I first opted for CrossRef API but later realized it did not yield the desired metadata. To resolve this, I explored alternatives and found that Semantic Scholar API, a free API, provided more precise data. Another challenge arose with DIFY's code block, which required all returns to be in dictionary format. This caused issues when processing JSON strings within the HTTP Request block, so I modified my approach by making an API call inside the code block for seamless data insertion into Neo4j. Additionally, due to frequent testing, I quickly exceeded my Hugging Face account's monthly limit, which I resolved by creating multiple accounts and using different API keys. Lastly, while integrating Neo4j with the DIFY workflow, I initially faced errors when attempting a local setup. However, by seeking assistance from online forums, I received valuable insights that led to a more effective solution. These challenges and their resolutions strengthened my problem-solving skills and enhanced the overall efficiency of the project.

3. What adjustments to your design and implementation were necessary during the implementation phase? What would you change or do differently if you had to do the portfolio task a second time? What would be potential areas for future improvement.

   Answer: During the implementation phase, I encountered challenges with my initial rough design, which required several adjustments to ensure the accuracy and effectiveness of the system. For instance, I initially attempted to create nodes and relationships using the "Cites" field. However, I found that the expected data was not available or sufficient for establishing meaningful relationships. To address this issue, I modified my approach by utilizing the "Fields of Study" field instead, which provided more relevant and structured data for creating relationships between nodes. These types of design refinements were essential in aligning the implementation with the available data, ultimately improving the overall structure and integrity of the knowledge graph. If I were to redo the portfolio task, I would incorporate a Citation node to enhance the representation of references within the knowledge graph. This addition would provide a more structured way to capture citation relationships between research papers, improving the accuracy of knowledge representation. Potential areas for future improvement include enhancing the knowledge graph structure by incorporating additional contextual relationships, such as co-authorship networks and journal impact factors, to provide deeper insights into research trends.

4. Include a brief section on ethical considerations when using these models in research domain.

   Answer: When integrating AI models into research workflows, it is essential to consider ethical implications such as data privacy, bias, and transparency. AI models rely on large datasets, which may contain sensitive or proprietary research information. Ensuring compliance with data protection regulations and maintaining researcher anonymity where necessary is crucial.

5. From the lecture/course including guest lectures, what topic excited you the most? Why? What would you like to learn more about and why?

   Answer: The topic that excited me the most from the lecture/course, including guest lectures, was Multiagent Systems. The concept of multiple agents interacting and collaborating to achieve shared goals, as well as dealing with distributed decision-making and problem-solving, is both fascinating and highly relevant in today's world. I am particularly interested in how these systems can be applied to real-world problems, such as in autonomous vehicles, smart cities, and complex simulations. I would love to learn more about the algorithms and frameworks that govern coordination, negotiation, and communication between agents, as well as how to design scalable and efficient multiagent systems for large-scale applications.

6. How did you find working with DIFY platform during the course work and portfolio exam? Would you recommend using DIFY in learning Generative AI technologies and why? What is the best start for learning Generative AI either by Python code or No-code platforms and why?

   Answer: Working with the DIFY platform during the coursework and portfolio exam was an insightful experience. The platform provided a user-friendly interface for integrating different components, such as API calls, code blocks, and graph databases, while offering flexibility for customizing workflows. I found its ability to combine no-code and low-code elements highly effective, allowing for rapid prototyping and testing of complex tasks without needing extensive coding knowledge. The integration with Neo4j was seamless, and the ability to automate data processing and visualization was particularly beneficial. I would highly recommend DIFY platform because of its flexibility. I think Python code is best because there I can do anything according to my choice, on the other hand, although there is some flexibility in No-code platforms, there are some limitations also. Therefore, I will prefer python code as a developer. For the beginners, No-code platforms are good enough.

7. How did you find the assignments and exercises in the course and how they help you in portfolio exam?

   Answer: I found the assignments and exercises in the course to be highly beneficial in preparing for the portfolio exam. The structured nature of the assignments allowed me to gradually build my knowledge and skills, progressing from basic concepts to more complex tasks. Therefore, I was able to solve this task for the portfolio exam successfully.

3 Conclusion

Through this task, I gained valuable experience in understanding how AI models process metadata to generate meaningful analytical outcomes. By working with LLMs, APIs, and graph databases, I explored how structured metadata can be extracted, analyzed, and utilized for knowledge representation. The process involved selecting the right tools, such as Semantic Scholar API for metadata extraction and Neo4j for visualization, which enhanced my ability to work with data-driven AI models. Additionally, integrating DIFY's code blocks and handling API constraints provided insights into workflow automation and optimization. This experience deepened my understanding of how AI models leverage structured data to derive insights, identify relationships, and enhance decision-making in research domains.

References

[1] Mistral AI. Mistral-7b instruct v0.3, 2025. Accessed: 2025-03-14.

[2] Semantic Scholar. Semantic scholar api, 2025. Accessed: 2025-03-14.

[3] Neo4j. *Neo4j HTTP API Documentation*, 2025. Accessed: 2025-03-14.

[4] Hugging Face. Hugging face, 2025. Accessed: 2025-03-14.