# Graph Databases and Applications with Large Language Models (NaLLM Project) - Homework #2

Sami Taider
84401097

November 18, 2024

## 1 Introduction

The NaLLM project by the Neo4j team explores the integration of Large Language Models (LLMs) with graph databases. The project aims to leverage Neo4j's graph database capabilities alongside LLMs to develop advanced use cases in Natural Language Interfaces, Knowledge Graph creation, and automated report generation. The repository contains code, blog posts, and various materials to demonstrate the synergies between these technologies.

The primary objectives of this homework were to:

1. Familiarize with the objectives of the NaLLM project.
2. Explore the repository's contents to understand the resources and components.
3. Investigate the Natural Language Interface to a Knowledge Graph.
4. Examine the Knowledge Graph creation methodology.
5. Explore the Report Generation components.
6. Provide a summary of findings, challenges, and questions.

This report will provide a comprehensive overview of these tasks, summarizing findings, challenges, and questions encountered during the exploration of the NaLLM project.

# Task 1: Familiarization with the Objectives of the NaLLM Project

The NaLLM project demonstrates the integration of LLMs with graph databases, focusing on three main use cases:

1. **Natural Language Interface to a Knowledge Graph:** This component enables users to query a Neo4j graph database using natural language, which is then translated into Cypher queries using LLMs.
2. **Creating a Knowledge Graph from Unstructured Data:** The project also demonstrates how to extract and structure data from unstructured text, converting it into graph data (nodes and relationships).
3. **Generating a Report:** Using static data and data retrieved from LLMs, the project automates the generation of reports that summarize and analyze data from graph databases.

# Task 2: Exploring the Repository's Contents

The main components of the repository include:

- **Backend Code (`api/`):** This folder contains multiple subdirectories and files that manage the functionality of LLM integration, data processing, and Cypher query generation :
  - Individual components such as `company_report.py`, `data_disambiguation.py`, `text2cypher.py`, and `unstructured_data_extractor.py`, handle specific tasks like data extraction, text parsing, and generating Cypher queries from text.
- **Frontend Code (`ui/`):** The frontend code is not explored in this report

- **Blog Posts and Documentation:** The project's progress and insights are shared through detailed blog posts. These posts cover topics such as fine-tuning LLMs, multi-hop question answering, and real-time graph analytics. These posts are useful for understanding the underlying challenges and methodologies explored during the project.
- **Demo Database:** The repository also includes a demo database running on `demo.neo4jlabs.com`. This database is connected to the project and serves for demonstrating the integration between Neo4j and LLMs.

# Task 3: Investigating the Natural Language Interface Component

The Natural Language Interface (NLI) is one of the core components of the NaLLM project. The NLI allows users to query a Neo4j database using natural language, which is then processed by LLMs and converted into Cypher queries. The goal is to enable users without knowledge of Cypher to interact with graph data using familiar language.
Key insights from the repository:

- The `text2cypher.py` file implements the core functionality of converting natural language queries into Cypher queries using an LLM.
- The system message generated in the `Text2Cypher` class is crucial in instructing the LLM to focus only on Cypher query generation.
- The LLM generates Cypher queries based on user input, which are then validated against the Neo4j database.

Challenges encountered:

- Ensuring that the LLM produces valid Cypher queries requires precise schema definitions and well-structured system messages. Any ambiguity in the schema or system message can result in incorrect or invalid queries.
- Handling complex queries that involve multiple relationships or nested data can be tricky, as it requires the LLM to correctly interpret the structure of the data.

## Task 4: Examining the Knowledge Graph Creation Methodology

The NaLLM project emphasizes the creation of Knowledge Graphs from unstructured data sources, such as text. This involves extracting relevant entities and their relationships, which are then converted into graph structures with nodes and relationships.
Key insights from the repository:

- The `unstructured_data_extractor.py` file provides the tools for processing unstructured text and generating graph entities.
- The `DataExtractor` class is central to converting unstructured text into graph structures, utilizing LLMs to identify key entities and relationships.
- The process systematically identifies nodes by their properties (such as name or occupation) and defines relationships by context (for instance, "Alice is a roommate of Bob").

Challenges encountered:

- Processing unstructured text can be unpredictable due to the lack of clear patterns in the data. This requires advanced natural language processing (NLP) models to accurately extract entities and relationships.
- Correctly mapping nodes and relationships is challenging when the data lacks context or when entities are ambiguous, making it difficult to draw accurate connections.

## Task 5: Exploring the Report Generation Components

The report generation component in the NaLLM project combines static data from the Neo4j graph database with dynamic data generated by LLMs to automate the creation of reports based on the graph's information.

Key insights from the repository:

- The report generation functionality is implemented across several components, with LLMs dynamically querying the graph and generating textual reports. The class 'SummarizeCypherResult' in the 'summarize_cypher_result.py' file plays a key role in this process by interacting with the LLM to generate concise answers based on Cypher query results.

- The combination of static data (stored in the graph) and dynamic data (generated by LLMs) allows for flexible and comprehensive reporting. For example, the 'VectorSearch' class in 'vector_search.py' facilitates querying the graph for relevant nodes based on a vector search, which can then be used to generate detailed reports.

- The report generation process involves querying the graph using Cypher queries (e.g., generated by the 'construct_cypher' function in 'vector_search.py') and using the LLM to structure and generate a narrative around the data. The 'generate_user_prompt' method of 'SummarizeCypherResult' constructs prompts for the LLM based on query results to guide the generation of reports.

Challenges encountered:

- Generating coherent and meaningful reports from complex graph data is challenging, especially when dealing with large datasets or many interconnected nodes and relationships. The system must ensure that data retrieved using Cypher queries, like those created in 'vector_search.py', is relevant and presented in an understandable way.

- Ensuring the accuracy of the generated reports and the relevance of insights requires careful design of the query templates and the system message. The 'system' message in the 'SummarizeCypherResult' class is an example of how the LLM is guided to generate concise and human-readable responses based on query results, which helps mitigate ambiguity in the generated content.

## Task 6: Summary of Findings, Challenges, and Questions

In summary, the NaLLM project provides a powerful integration of Neo4j with LLMs, demonstrating three core use cases: Natural Language Interface, Knowledge Graph creation from unstructured data, and automated report generation. The project leverages the power of LLMs to make graph databases more accessible and automate various processes.

Challenges encountered during the exploration include:

- Ensuring that LLMs generate valid and accurate Cypher queries.
- Dealing with the complexity of unstructured data and ensuring that the extracted entities and relationships are correct.
- Generating meaningful and accurate reports that combine static and dynamic data.

Future questions and areas for improvement include:

- How can the LLM's understanding of complex graph data be further improved to handle more intricate queries?
- What additional schema validation techniques can be implemented to ensure the LLM generates fully correct Cypher queries?
- How can the system be adapted to handle even more diverse types of unstructured data and generate more comprehensive knowledge graphs?