**THE UNIVERSITY OF MEMPHIS**

**Department of Computer Science**

**Querying Graph Databases using Natural Text**

**Nikhil Panda**

**07/05/2023**

**Project Advisor:**

**Dr.** **Xiaofei Zhang**

**Assistant Professor, Department of Computer Science**

**A project report submitted to the faculty of The University of Memphis in partial fulfillment of the requirements for the Master of Science Degree in the college of Arts and Science.**

# Abstract:

This project aims to provide a medium for all sets of users to access the Graph databases without learning any new graph querying language besides the natural text they are aware of. It is inspired from the fact that users can leverage modern day graph databases while making data-centric decisions and such tools are restricted to technical users alone due to limitation that every interested user needs to learn graph querying language to exploit the graph databases capabilities. So, we propose a web application, an end-user application to integrate graph databases using just natural language.

# Introduction:

Graph databases have emerged as powerful tools for representing and analyzing complex relationships in data. Unlike traditional databases based on the relational model, graph databases store data in nodes and relationships, providing a more intuitive representation of interconnected data. However, querying graph databases can be challenging, especially for non-technical users, they often require knowledge of query languages and database systems, limiting accessibility and hindering productivity. In this project, we aim to address this challenge by exploring the use of natural text to query graph databases.

By leveraging the capabilities of natural language processing (NLP) and graph database technologies, we seek to simplify the querying process, enhance user experience, and make graph databases more accessible to a wider range of users. By introducing natural text as an interface for querying, we aim to bridge this gap and enable users to interact with graph databases using familiar language patterns and expressions.

Our solution leverages the strengths of NLP models to process natural language queries and extract relevant information. We train a language model using existing NLP frameworks such as SpaCy or Hugging Face Transformers, fine-tuning it to understand the domain-specific context of our graph database. This training process involves labeling and annotating a dataset to teach the model to recognize entities, relationships, and keywords relevant to our use case.

By adopting a web application as our user interface, we enhance the user experience and provide a familiar environment for users to input their natural text queries. The web application can be designed using popular frameworks like Flask or Django, offering a responsive and intuitive interface that guides users through the query process. Additionally, we can incorporate features like autocomplete or query suggestion to assist users in constructing queries and improve their overall experience.

Overall, our solution aims to make querying graph databases more accessible and user-friendly. By leveraging natural text and NLP techniques, we simplify the query construction process, reduce the learning curve, and enhance the productivity of users. With this approach, users can harness the power of graph databases without needing to acquire in-depth knowledge of query languages or database systems.

# Background and Preliminaries:

Graph databases have gained popularity in recent years due to their ability to model and represent complex relationships among data entities. Unlike traditional relational databases, graph databases store data in nodes and relationships, offering a more intuitive representation of interconnected data. This structure enables efficient traversal and exploration of relationships, making graph databases suitable for scenarios with intricate data models.

In comparison to traditional databases, graph databases excel in handling complex relationships and traversing paths efficiently. Queries involving relationship patterns, such as finding paths between entities or identifying connected components, can be performed more easily and efficiently in graph databases. Traditional databases, on the other hand, often require complex join operations or recursive queries to achieve equivalent results.

Natural language querying introduces a new dimension to graph databases, enabling non-technical users to interact with complex data structures effortlessly. By allowing users to input queries in natural language, we tap into their existing knowledge and linguistic patterns, making the querying process more accessible and intuitive. Users can express their information needs using familiar language constructs, without the need to learn query languages or understand the underlying database schema.

However, there are challenges in enabling natural language querying for graph databases, especially for non-technical users. Constructing queries in natural language requires robust natural language understanding and processing capabilities. Ambiguity in language, varying levels of detail, and implicit context can introduce challenges in accurately interpreting and transforming natural language queries into executable database queries.

Query generation is a crucial step in the process of transforming processed natural language queries into executable database queries. We design rules and patterns to map the extracted information from the NLP model to appropriate Cypher syntax. These rules capture the semantics of the natural language queries and convert them into structured query patterns that can be executed on the graph database. Customizing these rules allows us to adapt the query generation process to specific business use cases and optimize the results.

To address these challenges, we employ NLP models and techniques. NLP models are trained to understand and process human language, enabling them to extract relevant information from natural language queries. We utilize pre-trained NLP models and fine-tune them on domain-specific data to capture the specific context of our graph database. This training process helps the models recognize entities, relationships, and keywords relevant to our use case.

To execute the generated Cypher queries and retrieve results from the graph database, we integrate our solution with a graph database management system such as Neo4j. We establish a connection to the database using a suitable driver and execute the queries using the provided API. The results are then processed and transformed into a suitable format for presentation to the user.

## Graph Database - Neo4j:

Neo4j is a leading graph database management system that is well-suited for the project of querying graph databases using natural text. It is a popular choice due to its strong support for graph data modeling and traversal, making it efficient in handling complex relationships. Neo4j's native graph storage and query language, Cypher, provides a powerful and intuitive way to interact with the graph database. With Cypher, we can easily express graph queries and patterns, which aligns well with the goal of enabling natural text queries. Neo4j also offers scalability, allowing the project to manage large datasets and accommodate future growth. Additionally, Neo4j provides a range of tools, libraries, and community support, making it a robust and well-documented choice for implementing the project successfully.

## Cypher:

Cypher is a Neo4j's native query language designed for querying and manipulating data in graph databases like Neo4j. It is a declarative language that allows users to express complex graph patterns and retrieve specific data from a graph database. Cypher provides a simple and intuitive syntax for working with nodes, relationships, and properties from the graph database.

**A diagram of a couple of people

Description automatically generated**

To generate Cypher queries from the processed natural language queries, we employ rule-based query generation techniques. We define rules and patterns that map the extracted information from the NLP model to appropriate Cypher syntax, ensuring the generated queries accurately represent the user's intent. These rules can be customized based on specific business use cases, allowing us to tailor the queries to the requirements of our graph database.

## Base Models for Natural Language Processing:

Utilizing already developed small language models like SpaCy brings several advantages. First, it saves time and effort compared to developing a new NLP model from scratch. These pre-trained models have undergone extensive training on large corpora and are ready to be used for various NLP tasks, including generating Neo4j Cypher queries.

Furthermore, these models offer reliable performance as they have been refined and assessed by the NLP community. They have been optimized for accuracy and efficiency, allowing for faster processing of natural language queries. The extensive language support provided by these models ensures compatibility with different languages and enables users to query the graph database using their preferred language.

The flexibility of small language models like SpaCy is another advantage. They can be easily fine-tuned and customized for specific domain contexts or use cases. This flexibility allows for the incorporation of domain-specific vocabulary, entity recognition, and query templates, tailoring the NLP model to the requirements of the project. Additionally, models like SpaCy come with a rich feature set that includes functionalities like tokenization, part-of-speech tagging, and dependency parsing. These features enable accurate understanding and processing of natural language queries, facilitating the extraction of relevant information for query generation.

In summary, leveraging already developed small language models like SpaCy offers time-saving benefits, reliable performance, extensive language support, flexibility for customization, a rich feature set, and access to a supportive NLP community. These advantages make these models a suitable choice for the project, enabling efficient and effective generation of Neo4j Cypher queries from processed natural language queries.

# Implementation Process and Solution:

In the context of this Use case, the implementation process can be broken down into the following parts:

## 1. Web Application:

The web application serves as the user interface for accessing the graph database. It provides a platform for users to input their natural language queries and interact with the system. The web application can be built using frameworks like Flask and Javascript library Vis.js, offering a responsive and intuitive interface. It should include features such as form inputs, query submission, and result visualization.

## 2. NLP Model Training:

The NLP model training process for this project involves several steps to fulfill the requirements and enhance the model's understanding of the context and entities relevant to the graph database.

Initially, small language models like SpaCy are chosen as the base models for training due to their time-saving benefits, reliable performance, extensive language support, flexibility, rich feature set, and community support. The specific models employed for training include en\_core\_web\_sm, en\_core\_web\_lg, and en\_core\_web\_trf, with the finalized en\_core\_web\_lg model selected based on available computing resources and desired feature set.

A screenshot of a computer

Description automatically generated

To customize the model for the project's use case, additional training is performed to improve entity recognition. This involves training the model to identify movie names as the 'WORK\_OF\_ART' entity and actor names as the 'PERSON' entity. By extracting movie names and training the model on them, the base model can accurately recognize movie titles in natural language queries. Similarly, training on actor names allows the model to identify and classify actors as entities.

Special keywords and labels are also marked as new entities to aid in understanding natural text and query generation. These additional entities help capture specific context and improve the accuracy of entity recognition and query understanding. To enhance the model's understanding of the context, Parts of Speech (POS) tagging, and Dependency Parsing techniques are utilized. By analyzing the POS and the dependency relationships between tokens in the text, the model can identify the roles and connections between words. This information is valuable for creating new entities, recognizing node names, and understanding relationship names in the natural language queries.

Throughout the NLP model training process, careful consideration is given to the available computing resources, the desired feature set, and the specific requirements of the project. By fine-tuning the base models and training them on domain-specific data, the NLP models can effectively understand the context and entities related to the graph database, enabling accurate processing and query generation from natural language inputs.

The trained NLP model becomes a crucial component in the overall project implementation, serving as the bridge between natural language queries and the generation of corresponding Cypher queries for execution on the Neo4j graph database. Its training encompasses entity recognition, context understanding, and query customization, ensuring that the model aligns with the specific requirements of the project and enhances the user experience of querying the graph database using natural text.

## 3. NLP Model Processing and Query Generation:

After the NLP model is trained, it plays a crucial role in the natural language processing (NLP) tasks to extract entities, keywords, and other relevant information from the user's natural language queries. The trained NLP model processes the queries and identifies important labels, keywords, and entities. This information is then extracted and used to generate structured queries in a format that the graph database can understand, such as Cypher queries.

To consume the trained NLP model, the model is utilized to identify the entities, nodes, and relationships that the user is interested in. These identified components serve as the basis for constructing query templates. The template attributes are derived from the processed natural language input, representing the key elements needed to generate meaningful Cypher queries.

Using a query generation process, the structured query is generated from the natural language input using identifies entities, labels, and keywords followed by parsing from rules created for query generation. This involves mapping the extracted information to appropriate Cypher syntax and patterns. In some cases, predefined rules can be created to convert the parsed natural language into Cypher queries, capturing the semantics and intent of the user's query.

A screenshot of a computer

Description automatically generated

Depending on the specific requirements of the use case, attributes and values can be customized to narrow down the scope of the results or search for keywords across all attributes. This flexibility allows for the generation of tailored queries that align with the desired outcomes.

For example, consider a natural language query like "Which are John's movies?" This query can be transformed into a Cypher query using the NLP model and query generation process. The result could be a Cypher query like "MATCH (p: Person)-[:ACTED\_IN]->(m:Movie) WHERE p.name = 'John' RETURN m." This query retrieves movies in which a person named John has acted and returns the corresponding movies.

By leveraging the trained NLP model, the project enables the conversion of natural language queries into structured and executable Cypher queries. This process bridges the gap between user-friendly natural language input and the requirements of the graph database, allowing users to interact with the graph database using familiar language constructs.

## 4. Cypher Generation and Neo4j Execution:

In this step, the processed information from the NLP model is used to generate a Cypher query. The Cypher query is then executed on the Neo4j graph database. The query is processed by the Neo4j engine, which traverses the graph, retrieves the relevant data based on the query, and generates the result. The result is then passed back to the web application for presentation to the user. This step involves establishing a connection to the Neo4j database, executing the Cypher query, and retrieving the results in a suitable format.

By following this implementation process, the project can provide a seamless and user-friendly experience for querying the graph database using natural language, leveraging the power of NLP and Neo4j to enhance data exploration and analysis.

A diagram of a software model

Description automatically generated

Throughout the implementation process, it is important to ensure proper validation and cypher query generation in less time. The web application should provide a smooth and intuitive user experience, guiding users through the process of entering queries, processing them using the NLP model, executing the Cypher query on the Neo4j database, and presenting the results in a user-friendly format.

# Experiment Details:

In this experiment, we aim to demonstrate the effectiveness of using small NLP-trained models to generate executable Cypher queries on a Neo4j database for retrieving results in a use case involving a small movie database. The objective is to display how natural language queries can facilitate finding connections between individuals based on their works and mutual participation.

## Experimental Setup:

i. **Data**: We utilize a small movie database stored in a Neo4j database. The database contains information about movies, actors, and their relationships.

A screen shot of a diagram

Description automatically generated

A screenshot of a computer program

Description automatically generated

ii. **NLP Model**: A small language model like SpaCy’s en\_core\_web\_lg has been trained and fine-tuned on domain-specific data to understand movie-related entities, attributes, and relationships. The model has been customized to recognize movie names as "WORK\_OF\_ART" and actor names as "PERSON" entities. It also considers special keywords and labels to aid in query understanding. Base model can be selected based on our business requirements and the rich feature set it can offer besides the pipelines it could offer for various NLP tasks. The Spacy’s en\_core\_web\_lg is offering all these pipelines for various NLP tasks like parts of speech tagging, dependency parsing etc…, to easily understand the context of the user’s natural text.

A diagram of a diagram

Description automatically generated

iii. **Web Application**: A web-based user interface has been developed to accept natural language queries from users and display the results obtained from the Neo4j database.

## Experiment Procedure:

i. **User Inputs:** Users are prompted to input their natural language queries related to finding connections between individuals based on their works and mutual participation in the movie database. They can use queries such as "Find actors who have worked together in movies" or "List out the Actor A’s movies."

ii. **NLP Processing:** The NLP model processes the natural language queries and extracts the relevant entities, keywords, and relationships. It utilizes the trained model's capabilities to understand the context and generate structured representations of the queries.

iii. **Cypher Query Generation:** Based on the extracted information, the NLP model generates corresponding Cypher queries. The queries are constructed using predefined rules and templates that map the natural language input to the Cypher syntax. The queries capture the relationships, attributes, and conditions specified in the natural language queries.

iv. **Cypher Query Execution:** The generated Cypher queries are executed on the Neo4j database. The queries traverse the graph, retrieve the relevant nodes and relationships, and fetch the desired results. The database engine processes the queries efficiently to optimize query performance.

v. **Result Presentation:** The obtained results are presented to the users through the web application, which uses Javascript library Vis.js. The web application fetches and displays the data in a user-friendly format, displaying the connections between individuals based on their works and mutual participation. The results can include actor’s works, movie associations, and other relevant information.

\*\* Entire model training has been done in Google Collab due to computation limits. Once Model training is completed, we serialized the model using Pickle and stored it in our local machine. A backend service has been built using Flask and python to reload the trained NLP model locally and then to process the NLP information before producing cypher queries using NLP processed response and parsing rules.

## Steps to run this Project:

1. Run the app.py under Project after installing flask using pip.

pip install flask

python app.py

1. Run and open the index.html in web browser (preferably Chrome).
2. Provide the natural text you wanted to query from the database.
3. Response will be presented on the web page using Vis.js and configuration passed to it.

If we ever trained the NLP model again, we would save that one using pickle and reload the new one from project’s ‘query\_processing.py’

## Evaluation and Analysis:

The accuracy and relevance of the retrieved results can be evaluated by comparing them with the user’s expectations and domain knowledge. The effectiveness of the NLP model and the generated Cypher queries is assessed based on their ability to correctly capture the intended results. User feedback can also be collected to gather insights on the usability and satisfaction of the natural language querying system.

# Results

## Expected Results and Outcomes:

The experiment aims to demonstrate that using small NLP-trained models to generate executable Cypher queries for querying a Neo4j database based on natural language inputs can effectively retrieve relevant results. The expected outcomes include:

* Successful retrieval of connections between individuals based on their works and mutual participation.
* Accurate recognition of movie names and actor names, enabling precise query generation.
* Efficient execution of Cypher queries on the Neo4j database, ensuring reasonable query response times.
* User-friendly presentation of the results in the web application, facilitating easy interpretation and understanding of the connections.
* Positive user feedback indicating the usefulness and effectiveness of the natural language querying system.

## Actual Outcomes:

The project of querying graph databases using natural text has yielded promising results for easy to moderate level queries. Users can effortlessly interact with the graph database using natural language queries, benefiting from the power of graph databases and the simplicity of natural language processing. However, it is important to acknowledge the need for robust training to handle more complex queries effectively. While natural text to query graph databases brings numerous benefits, there are also potential limitations and criticisms. Challenges like ambiguity in natural language queries or potential performance issues may arise.

A screen shot of a computer

Description automatically generated

A screenshot of a computer

Description automatically generated

While testing the entire NLP and query generation process, 7 out of 10 questions have produced promising results while the other 3 cannot. Around 73% of accuracy has been observed in this process and evaluation is manual process by verifying the output using my knowledge in Cypher and database. There might be unexplored cases where complex queries could not generate a promising result, and few could. So, the observed accuracy is based on the instances we explored and evaluated for this project.

A white screen with black text

Description automatically generated

#### NLP Process Limitations:

During the experimentation phase, certain limitations were observed in the NLP process. Specifically, the NLP model struggled to accurately identify the context for keywords such as "1990s" and "co-actors." These keywords require explicit handling to ensure proper recognition and understanding. Further refinement and fine-tuning of the NLP model can help address these limitations, improving the accuracy and effectiveness of the natural language processing stage.

#### Web Application Challenges:

One of the challenges identified in the web application component is the limited interaction capabilities within the graph visualization using Vis.js. While Vis.js provides a convenient library for graph visualization, it may require additional libraries or custom development to enhance interactivity and functionality. This limitation can be addressed by exploring other graph visualization libraries or incorporating custom development to provide a more interactive user experience.

#### User Feedback and Evaluation:

To assess the effectiveness of the project, user feedback and evaluation were conducted. Users were asked to provide their feedback on the usability, intuitiveness, and overall experience of using natural language queries to interact with the graph database. Their input played a valuable role in identifying areas for improvement and understanding the practicality and usability of the natural language querying system.

## Steps to Replicate the application on other databases:

A diagram of a software development process

Description automatically generated

### Base Language Model Selection:

Choose a base language model that aligns with your requirements and the specific pipelines you need. Consider factors such as the entities and labels you want the model to recognize, the language support required, and the available pipelines, features, and functionalities. It is essential to select a model that provides the necessary capabilities to process natural language queries and generate appropriate Cypher queries.

### NLP Model Training:

Once you have chosen the base language model, train it to fulfill your desired business requirements. This step involves fine-tuning the model using domain-specific data and incorporating any additional entity names or labels that are specific to your database. By training the model on relevant data, you can enhance its understanding of the context and improve its accuracy in recognizing entities and relationships.

### Processing from NLP response:

After training the base model, utilize the NLP capabilities to parse the natural language information provided in the user queries. The model will identify the nodes and relationships mentioned in the queries, extracting the necessary information for query generation. This parsing process involves analyzing the structure, entities, and keywords present in the natural language queries and transforming them into a format that can be used to generate Cypher queries.

### Cypher generation from Business rules:

With the parsed NLP information, create your own rules and logic to generate Cypher queries. These rules should capture the semantics and intent of the natural language queries and translate them into appropriate Cypher syntax. The rules can include mapping specific phrases or keywords to predefined query templates, defining relationship patterns based on the context, and customizing attribute values to narrow down the query results.

### Customize Web Application:

To provide a better user experience and cater to your specific business cases, customize the web application through which users interact with the database. Enhance the user interface to facilitate the input of natural language queries and optimize the presentation of the query results. Consider incorporating additional features, such as autocomplete or suggestion functionalities, to assist users in constructing queries more efficiently. Tailor the application to meet the specific needs of your database and the users interacting with it. Amend the configuration passing into Vis.js instance to reflect new Node, Relationships names besides the interactions your application needs.

By following these steps, you can replicate the process of querying graph databases using natural text on other databases. This approach allows you to leverage the power of NLP models and natural language queries to simplify the interaction with your database, improve user experience, and facilitate the exploration and analysis of complex data structures. Keep in mind that each database may have its own specific requirements and considerations, so it is important to adapt these steps accordingly to ensure compatibility and optimal results.

# Conclusion

This project demonstrates the power and potential of using natural text to query graph databases, providing users with an efficient and intuitive way to access and analyze complex data. By combining the strengths of graph databases and natural language processing (NLP), users, including non-technical individuals, can interact with graph databases effortlessly, gaining valuable insights and discovering meaningful relationships within the data. The ability to visualize query results in a user-friendly interface further enhances the user experience, enabling users to explore and understand data more effectively while making data-driven decisions. While there may be challenges in query construction and optimization, this project highlights the accessibility and practicality of leveraging natural language and graph databases for various real-world applications.

### Real-World Use Cases:

#### Healthcare Analysis:

The project can assist in healthcare analysis, where the complex relationships among patient records, medical conditions, treatments, and research need to be analyzed. Natural language queries like "Find patients with a specific condition and their related treatments" can be transformed into Cypher queries to retrieve relevant data from the graph database. By visualizing medical relationships, such as patient-doctor interactions, treatment effectiveness, and disease patterns, healthcare professionals can gain valuable insights to improve patient care, conduct research, and make informed decisions.

#### Knowledge Graph Exploration:

The project enables users to explore and navigate large knowledge graphs, where vast amounts of interconnected information exist. Users can input natural language queries like "What are the connections between topic A and topic B?" to explore the relationships and connections within the graph database. The web application allows users to visualize the results, facilitating a better understanding of complex knowledge structures. This use case is particularly beneficial for domains like education, research, and data analysis, where users need to uncover relationships, trends, and patterns within extensive knowledge repositories.

# Future Work

* Improve the NLP training for robust natural text processing to process complex queries.
* Implementing attribute mappings for clumsy attribute names would help narrow down the results.
* Training the NLP model on Cypher queries dataset may ease the NLP process and dependency.
* Offer autocomplete or suggestion features to assist users in constructing queries in the web application.
* Also, user feedback enhances the user experience.

# References

[1] [Natural Language Processing (NLP) for Business Use-Cases](https://alphabytesolutions.com/natural-language-processing-nlp-for-business-use-cases/)

[2] [The Graph Database Advantage for Enterprise Architects](https://www.avolutionsoftware.com/abacus/the-graph-database-advantage-for-enterprise-architects/)

[3] [A Natural Language Interface for Querying Graph Databases](https://dspace.mit.edu/bitstream/handle/1721.1/119708/1078222310-MIT.pdf?sequence=1)

[4] [Cypher Query Language](https://neo4j.com/developer/cypher/)

[5] [NLDS-QL: From natural language data science questions to queries on graphs: analysing patients conditions & treatments](https://www.researchgate.net/publication/362858884_NLDS-QL_From_natural_language_data_science_questions_to_queries_on_graphs_analysing_patients_conditions_treatments)

[6] [Natural Language to SQL from Scratch with Tensorflow](https://towardsdatascience.com/natural-language-to-sql-from-scratch-with-tensorflow-adf0d41df0ca)

[7] [Accelerating Towards Natural Language Search with Graphs](https://neo4j.com/blog/accelerating-towards-natural-language-search-graphs/)