From Natural Language to Knowledge Graph: An LLM-Driven Approach to Automated Query Generation and Evaluation

A pipeline for syntactic compliance and semantic alignment in LLM-generated Cypher queries

Author: Thijmen van der Meijden

First examiner: Dr. Michael Behrisch
Second examiner: Prof. Dr. Massimo Poesio

A thesis presented for the degree of
Bachelor of Science, Kunstmatige Intelligentie

Faculteit Geesteswetenschappen
Universiteit Utecht
Nederland

Faculty of Humanities
Utrecht University
The Netherlands

31-01-2025

Abstract

Knowledge graphs are powerful tools for data representation and retrieval, yet, their complexity often restricts their use to experts. This thesis proposes a Large Language Model Query Recommender pipeline to bridge this gap by automating the generation and evaluation of meaningful queries for knowledge graphs. The pipeline interprets user intents, converts them into structured queries, and evaluates these queries for both syntactic correctness and semantic alignment with the user's request. A key focus of this work is the development of automated evaluation methods to ensure the quality of generated queries without human intervention. By automating both the generation and evaluation processes, this thesis seeks to democratize access to knowledge graphs, making them more user-friendly and accessible to non-experts. The findings contribute to improving the usability and reliability of complex data structures, enabling broader audiences to effectively explore and utilize their potential.

Contents

1 Introduction

Knowledge graphs represent vast, interconnected webs of information that reveal the relationships among entities like people, places, objects, and concepts. Their structured nature allows them to answer complex queries by navigating these relationships. However, the intricacies of knowledge graphs often render them accessible only to domain experts familiar with specialized query languages such as Cypher. This limitation restricts the broader usability of these powerful tools.

Recent advancements in Large Language Models (LLMs), such as GPT-4, Llama 3.2, or the heavily discussed DeepSeek-R1, have demonstrated their ability to interpret natural language inputs and generate contextually relevant outputs, including computer code and queries. **This research investigates the feasibility of leveraging LLMs to bridge the accessibility gap in interacting with knowledge graphs.** By automating the generation of meaningful, schema-compliant queries from natural language inputs, this study aims to make knowledge graphs more user-friendly for non-experts. Additionally, it explores methods to evaluate these queries automatically, ensuring their accuracy and relevance without human intervention.

1.1 What are knowledge graphs?

Knowledge graphs are structured representations of information, where entities (nodes) are connected by relationships (edges). For example, a node representing an actor, like Tom Hanks, might connect to a node for a movie, like Cast Away, through an `ACTED_IN` relationship. An example of a knowledge graph can be seen in figure 1. These structures facilitate complex queries, enabling users to explore data in ways that would be difficult with traditional databases.

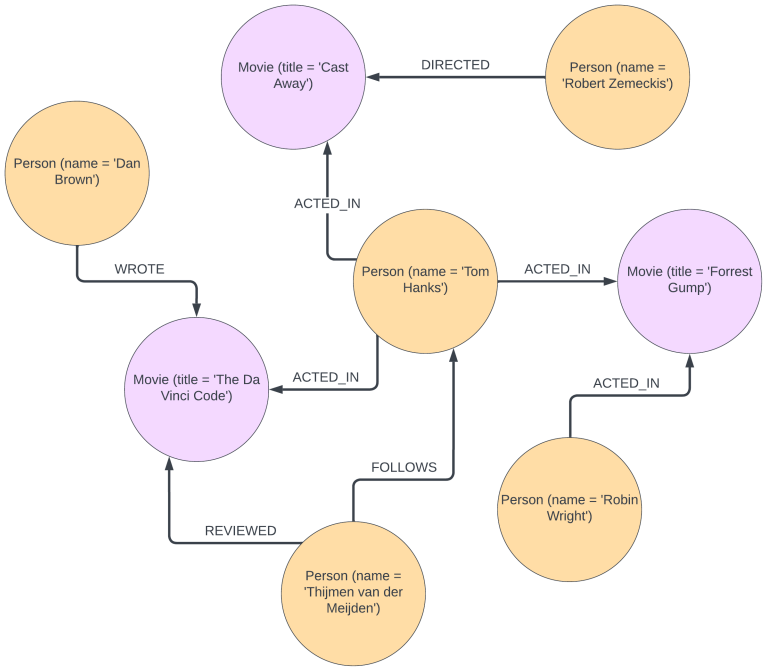

Figure 1: Example of a knowledge graph.

Imagine a knowledge graph as a massive, interconnected web in a library where books (entities) are linked by bookmarks (relationships). Each bookmark describes how two books relate—for instance, one book might reference another as a source, or they might share the same author or topic. A person's "book" could be linked to all the projects they've worked on or the people they've collaborated with. This web allows you to trace connections through various paths—like following a trail of references or exploring all works connected to a single topic.

While powerful, making a query for a knowledge graph typically requires expertise in query languages such as Cypher. For instance, a user wanting to find all movies starring Tom Hanks would need to write:

```
MATCH (a:Person)-[:ACTED_IN]->(m:Movie)
WHERE a.name = "Tom Hanks"
RETURN m.title
```

Such technical requirements make knowledge graphs inaccessible to many potential users. For a non-technical user, translating their intent into such a structured query can be intimidating and error-prone.

1.2 The role of Large Language Models

LLMs have shown remarkable capabilities in understanding and generating natural language text. Their potential extends to translating user intents expressed in plain language into structured formats, such as queries for knowledge graphs. For instance, given the user input and the corresponding knowledge graph schema *'Show me all movies starring Tom Hanks'*, an LLM could possibly generate the corresponding Cypher query.

These models excel in tasks like interpreting intent, summarizing text, and even generating code snippets. By leveraging these strengths, we can bridge the gap between non-technical users and complex data systems. For example, an LLM can not only generate a query but also provide explanations or suggest refinements based on the schema's structure.

However, this process is not without challenges. **Ensuring the generated query is both syntactically correct and semantically aligned with the user's intent is critical because otherwise it is not a meaningful query.** Additionally, the query must comply with the schema of the knowledge graph, which defines the permissible nodes, edges, and attributes. A small deviation from the schema could render the query invalid, making precise alignment with the knowledge graph's structure important.

1.3 Challenges in query generation and evaluation

Translating natural language into structured queries involves two key challenges:

- **Syntactic correctness:** The query must adhere to the grammar of the query language and the schema of the provided knowledge graph.
- **Semantic correctness:** Even if syntactically correct, a query may fail to capture the user's intent. For example, a query might return *all* the movies in the knowledge graph when the user *only* asked for those released in 2020.

Traditional methods of evaluating queries on semantic correctness rely on human intervention, which is subjective and not scalable. **Automating this evaluation process is a core objective of this research**, focusing on developing schema-aware validation, LLM-based self-checking, and re-interpreting semantic similarity analysis methods. These automated techniques aim to remove the dependency on manual assessments while maintaining high standards of accuracy and relevance.

To ensure semantic correctness, two methods will be employed.

- **Self-checking:** We can use a second "referee" LLM (the same LLM in a blanco state) to review the generated query, comparing it against both the user's request and the schema, thus offering a semantic "sanity check".

- **Re-interpretation:** Another form of evaluation involves turning the query back into plain English and comparing that version to the user's original request using vector embeddings. If the paraphrased statement closely aligns with what the user asked, it is more likely the query itself is accurate.

Additionally, LLMs themselves are not immune to errors. Issues such as "hallucinations", where the model generates plausible but incorrect outputs, pose challenges to ensuring reliability. **Because LLMs are, in essence, the same machines that generate the query, their role in evaluating their own outputs can be limited.** They might reproduce their own biases or errors. Especially with the self-checking method there is a risk of the model agreeing with itself. To mitigate this, the external metrics—such as embedding-based similarity scoring (for example, embeddings for the re-interpretation)—and a schema compliance method (which parses the query and matches it against the given schema) also ensure syntactic correctness. By combining LLM-based and algorithmic checks, the pipeline gains a multi-layered approach to verification, hoping to catch a broader range of errors.

1.4 Research question & hypothesis

The central research question guiding this thesis is:

> *How can the quality and relevance of queries on knowledge graph generated by Large Language Models be automatically assessed and implemented, ensuring meaningful outputs aligned with the schema and user request without human supervision?*

Leveraging LLMs to generate and evaluate queries for knowledge graphs can significantly improve accessibility for non-expert users by automating the translation of natural language inputs into syntactically and semantically correct queries. Specifically, this thesis hypothesizes that the re-interpretation semantic **similarity score will have a high correlation ($p < 0.05$) with semantic correctness.** Furthermore, this thesis hypothesizes that the LLM-based **self-checking will achieve at least 80% agreement with manual evaluations**, providing a reliable automated pipeline for query assessment.

1.5 Broader implications and contributions

The implications of this research extend beyond making knowledge graphs accessible. By reducing the technical barriers associated with querying complex data systems, **this work has the potential and goal to democratize data-driven decision-making.** For instance, researchers, educators, and even hobbyists could explore intricate datasets without needing

advanced technical training. This aligns with broader trends in AI and machine learning aimed at enhancing usability and inclusivity.

Furthermore, this research contributes to the growing field of human-AI collaboration. By enabling LLMs to act as intermediaries between users and complex systems, it paves the way for more intuitive interfaces and tools that bridge the gap between natural language and technical operations.

Specific contributions in this work include the novel semantic evaluation approach, which combines LLM-based self-checking and re-interpretation through semantic similarity analysis. This is a unique contribution because no prior work has implemented a comprehensive pipeline that automates the evaluation of LLM-generated knowledge graph queries with such a multi-layered approach.

2 Related works

AI has been a hot topic of discussion for years now. The integration of LLMs and knowledge graphs represents a promising avenue for enhancing data exploration and accessibility. Existing research has focused on leveraging knowledge graphs to enrich LLMs with external knowledge, enabling improved reasoning capabilities and more accurate, context-aware responses. However, **the "reverse"—using LLMs to make querying knowledge graphs more accessible to non-expert users—remains underexplored.** This gap highlights the importance of investigating how LLMs can facilitate intuitive interactions with knowledge graphs while trying to maintain semantic and syntactic accuracy.

2.1 Using knowledge graphs to improve AI

Knowledge graphs have gained prominence as tools to enhance AI models, particularly in the realm of explainability and factual grounding. Futia and Vetrò [1] argue that knowledge graphs, due to their transparent structure and semantic clarity, can help elucidate how deep learning systems make decisions—an important feature in sensitive domains like healthcare. Abu-Rasheed et al. [2] further highlight how knowledge graphs serve as sources of factual context to mitigate LLM "hallucinations," thereby improving model reliability. However, these works fundamentally aim to enhance models using knowledge graphs. In contrast, **this thesis seeks to use LLMs to democratize access to knowledge graphs themselves**, thereby reversing the typical direction of this integration.

2.2 Automation of evaluating LLM outputs

Ensuring the correctness and reliability of LLM-generated content is an ongoing challenge. Van Schaik and Pugh [3] focus on evaluating LLM outputs in text summarization, underscoring the inadequacy of traditional metrics like BLEU or ROUGE when applied to more complex generative tasks. Desmond et al. [4] propose *EvaluLLM*, in which an LLM assists in evaluating its own outputs to reduce human effort. Though these studies advance scalable evaluation frameworks, they primarily target text summarization. In contrast, **the evaluations in this thesis tackle structured queries for knowledge graphs**, requiring verification of both syntactic correctness and semantic alignment with a user's intent—an area not addressed by these existing frameworks.

2.3 Manual knowledge graph evaluation

Balcioglu [5] explores how powerful LLMs (e.g., GPT-4 or DeepSeek) can generate natural language responses for knowledge graph queries. While this approach simplifies query formulation for end-users, it relies on manual human judgments to determine whether the generated query or answer is correct—introducing subjectivity and limited scalability. In contrast, **this thesis focuses on automating the evaluation process**—from checking the query's schema compliance to assessing its semantic fidelity via LLM-based self-checks and re-interpretation methods. By minimizing human involvement, this work aims to create a robust, scalable pipeline for generating *and* validating knowledge graph queries.

3 LLM-based query suggestion framework

This section outlines the step-by-step process of the pipeline I have designed to generate queries for knowledge graphs based on user-provided natural language input. The code of this entire pipeline can be visited at: https://git.science.uu.nl/graphpolaris/experiments/llm-query-recommender. It is called *LLM Query Recommender*

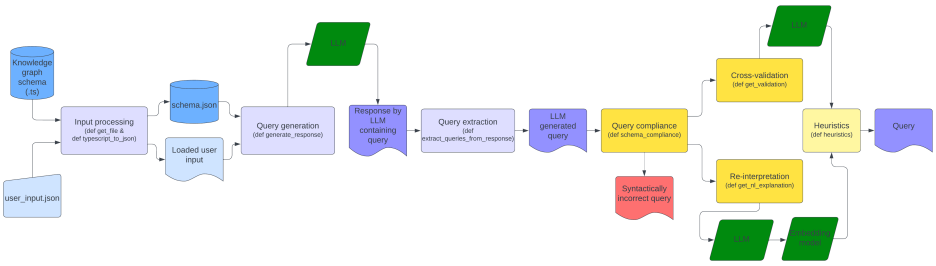

Figure 2: Visualization of the total pipeline. This is the engineering and research part combined.

3.1 Engineering

This subsection focuses on the engineering aspect of the pipeline and is not an unexplored topic. This is a necessary step to get to the research part, and it is crucial for those who want a broken-down explanation of the code. The first part of the pipeline is visualized in figure 3. As an input, this part needs a knowledge graph schema Typescript file in which the schema itself is in JSON format. The output of this "engineering" part is one LLM generated query.

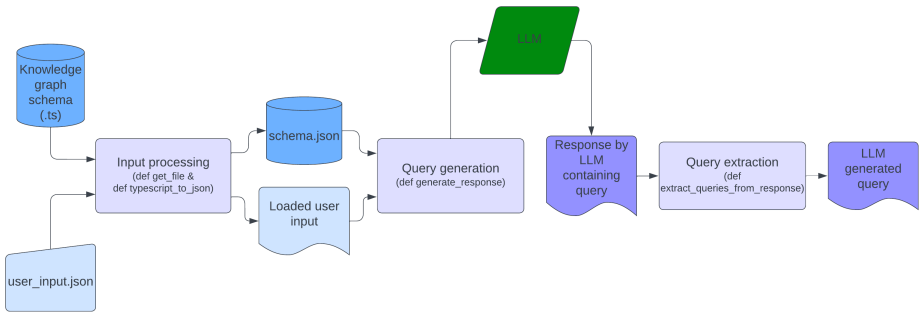

Figure 3: Visualization of the "engineering part" of the pipeline.

1. **Input handling** The pipeline begins with an input which consists of:

- **Natural language query:** A user request in plain language, such as "Show me all directors who directed a movie in 1990."
- **Knowledge graph schema:** A schema file defining the nodes, edges, and attributes in the knowledge graph. The schema is provided as a TypeScript file and is loaded into a JSON format to ensure compatibility with subsequent processing steps.

Figure 5 and 6 show an example of how a schema is represented in JSON. A visualization of a schema is visible in figure 4.

Key steps:

- **File handling:** The schema file path is specified in `'user_input.json'`, and the file is validated to ensure it exists.
- **TypeScript to JSON conversion:** The pipeline extracts the schema from the TypeScript file by identifying and removing the unnecessary TypeScript lines. What remains is the schema in JSON format. The schema is saved in a (new) file as schema.json.

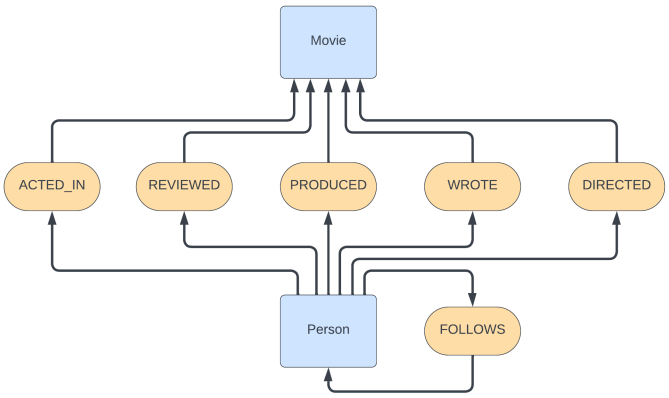

Figure 4: Visual representation of the Movie schema.

Figure 5: Example of a node in JSON format.

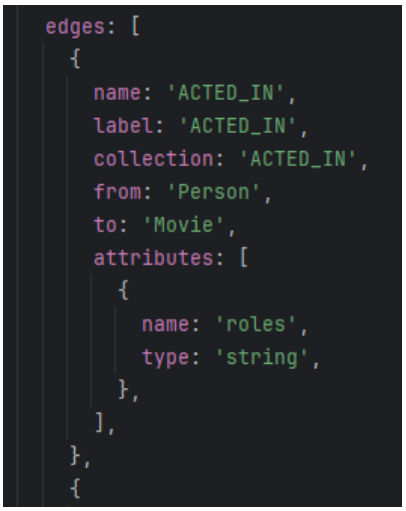

Figure 6: Example of an edge in JSON format.

2. **Query generation** The next stage involves generating a query based on the user request and the schema. This is achieved using a dedicated query generator class.
   Key steps:

   - **Prompt construction:** The Generator class constructs a prompt by combining the schema and the user's request. For example:

     `Given the following schema of a knowledge graph,`

```
generate 1 valid and working Cypher query (without
adding comments) that will comply with the request.
Mark the start of the query like this: ’‘‘‘cypher’,
and the end of the query like this: ’‘‘‘’ {schema}.
{request} (f.i.: "Show me all directors that
directed a movie in the year 1990.")
```

- **LLM query generation:** The prompt is passed to the LLM, which generates a response containing one (or potentially more) Cypher query(ies).

3. **Query extraction** Once the response is generated, the pipeline extracts the *first* generated query (if there are multiple) using indexes of the start and end of the generated query. The Evaluator class identifies and extracts Cypher queries enclosed in code blocks (e.g., “‘cypher ... “‘) from the LLM’s output

Having described the engineering pipeline, we now turn to the core research challenge: How to automatically evaluate semantic correctness.

3.2 Research

This subsection outlines the step-by-step process of the pipeline designed to evaluate generated queries. This section is mainly unexplored in the existing literature and is researched in this thesis. This part of the pipeline is visualized in figure 7.
This part takes as input a query in the language Cypher and outputs a ’similarity score’ as an integer between 0 and 1 and a binary ’Yes’ or ’No’.

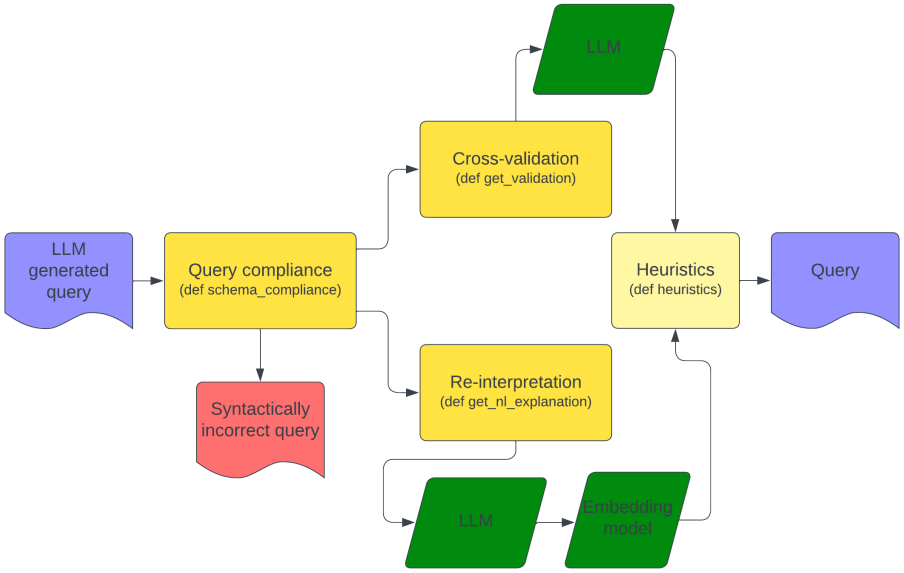

Figure 7: Visualization of the 'research part' of the pipeline.

1. **Query compliance** The query is checked for syntactic correctness by checking whether it follows the provided schema. This step is done before the other evaluation methods because if the query does not comply with the schema it can not be executed and thus is a bad query.
   Key steps:

   - **Regular expression matching:** Nodes, edges and attributes are extracted and put into variables from the query using Regular Expressions.
   - **Existence check:** Node, edge and attribute variables in the query are analyzed to confirm they exist in the schema.
   - **Query handling**: Only the first compliant query is saved in the scenario in which the LLM generates multiple queries in 1 response. So, other possible queries are disregarded.

   This process is explained in more detail in section 4.1.

2. **Query evaluation** To ensure the query aligns with the user's request, the pipeline evaluates it using self-checking and re-interpretation methods.
   Key steps:

   - **Self-checking**: The LLM is prompted to evaluate the query against the schema and the user's request, producing a binary

'Yes' or 'No' response. This process is explained in more detail in section 4.2.

- **Semantic similarity:** The pipeline uses the LLM to translate the query back into plain English. An embedding model compares the similarity between the sentence of the user request and the generated natural language explanation and generates produces a "similarity score". This process is explained in more detail in section 4.3.

All the results from these key steps are inputs for the following heuristic function.

3. **Heuristics** This function is making the final decision whether the semantic similarity is good enough and thus is likely accurate to the original user's request. This function will be disabled for the experimentation because it is not useful to filter low scores prematurely. If the pipeline will ever be deployed on a large scale, this function is necessary to give the user good advice.

3.3 Software and hardware environment

This entire project is developed in Python 3.12 with PyCharm 2024.3 (Professional Edition). The LLM used for the entire thesis is `Meta-Llama-3-8B-Instruct.Q4_0` [6]. The choice to use this model comes from 2 factors; it has been trained on code and some "pre-experimenting" showed most promise in this model out of several models tested that are able to run locally. The actual hardware used to develop and run the code and models is the following: AMD Ryzen 7 9700X (CPU), G.Skill DDR5 Flare X5 2x16GB 6000 MT (RAM) & NVIDIA GeForce RTX 4070 Ti SUPER (GPU). The LLM is run on the GPU using the NVIDIA optimised Kompute backend. The embedding model is `multi-qa-mpnet-base-cos-v1` [7] and runs on the CPU. The CPU has been water-cooled to ensure there is no thermal throttling. The operating system of the computer is Windows 11 Pro Version 24H2 Build 26100.2605.

3.4 Experiment

The experiment will be conducted by running the pipeline multiple times with different prompts, each ranked according to a level of complexity on 3 different schemas. The schemas can be seen in appendix A. The LLM is given a maximum context window of 4096 tokens, which allows for enough context memory for the LLM to not have to throw away parts of the context, which might influence results.

Prompts that contain numerous conditional statements—such as *'Find me all movies directed by Alex Johnson and reviewed by my friend Tyler2001,*

with a rating of at least 5'—are considered more difficult than simpler prompts with fewer conditions (e.g., *'Find me all movies directed by Alex Johnson'*). A prompt is rated a difficulty from 1 to 6. This difficulty represents how many conditions the request has, so difficulty 1 has one condition while difficulty 6 has six conditions. An example of each prompt difficulty can be seen in table 1.

Example of a prompt	**Difficulty**
Find all movies written by Ava Brown.	1
Find all movies where Olivia Taylor acted and were reviewed by Jordan54.	2
Find all movies written by Rachel Brown that were reviewed by Peter69 with a rating below 5.	3
Find all movies directed by Ethan Green, reviewed by Lucas88, released after 2010, and with more than 2000 votes.	4
Find all movies directed by John Farlee, written by Rachel Brown, reviewed by Nathan76 with a summary containing "mediocre," and fewer than 300 votes.	5
Find all movies directed by Lucas Brown, produced by Ava Johnson, written by Sophia Carter, and reviewed by Ethan3105 with fewer than 500 votes and a rating below 3.	6

Table 1

After a valid result is generated, each result is manually checked for correctness.

As previously mentioned in 3.3, the experiment will be run using a single local LLM on the same hardware. The prompt complexity and schema are the independent variables, while the similarity scores and boolean self-checking responses are the dependent variables. This analysis is crucial for assessing the effectiveness of these evaluation methods.

Actually, each difficulty level includes one more condition than its numerical difficulty. For example, consider prompt 1 in Table 1: it specifies that the query must be a movie (condition 1) and that it must be written by Ava Brown (condition 2). This approach was chosen to prevent difficulty levels with only a single condition from becoming overly constrained by the number of different node types available in the schema. Without this adjustment, for the Movie schema, only two prompts would be possible: 'Find all persons' and 'Find all movies.'

3.5 Extra prompt engineering experiment

An additional experiment will be conducted to evaluate the impact of prompt phrasing and context on the pipeline's performance. Using the Movie schema, the pipeline will be tested with a baseline prompt—"Find the movie 'Cast Away'"—as well as multiple variations of this prompt. Each variation will maintain the same objective but differ in wording, structure, or the level of context provided to the LLM. A prompt that has no request but is an iconic line from the movie is also added to test the extremes. The purpose of this experiment is to assess how the pipeline responds to changes in prompt design and whether certain phrasing or contextual cues improve the accuracy, efficiency, and semantic alignment of the generated queries.

4 Evaluation of generated queries

How can the quality and relevance of the queries generated by the LLM be assessed, and what metrics or tools can be utilized for this evaluation? A central challenge in automatically evaluating generated Cypher queries is **ensuring that they accurately capture a user's original request.**

4.1 Schema compliance

Ensuring schema compliance is a critical step in evaluating the quality of LLM-generated queries. Schema compliance refers to the correctness of the query with respect to the structure defined by the schema of the knowledge graph. This includes verifying the proper use of node names, edge names, attributes, etc.

A compliant query must not only be syntactically valid but also conform to the logical structure of the underlying graph. For instance, if the schema specifies that a 'Person' node is connected to a 'Movie' node through an `'acted_in'` relationship, the query must not introduce a non-existent or invalid relationship type (e.g. `'played_in'`). Non-compliance in these areas can lead to errors during query execution or the retrieval of irrelevant or misleading results.

To ensure automated schema compliance, an automated evaluation process can be applied. **This process parses the generated query by matching Regular Expression patterns and self-checks it against the schema to identify inconsistencies.** Such 'basic' evaluation mechanisms are essential for preventing errors, ensure syntactic correctness, and are the first step of being a meaningful query. By prioritizing schema compliance, the pipeline can ensure that queries are both functional and aligned with the intended structure of the knowledge graph, forming a good foundation for later evaluation steps. If queries are not conform the schema's or query language's rules , they are discarded because doing more evaluation on them does not fix the structural issue. Looking at the syntactic correctness is a quick step while semantic correctness is a time-consuming process, so that is why this evaluation step is done first in this pipeline.

Below, the individual validation steps are outlined.

4.1.1 Node validation

Nodes are extracted from the query using the pattern:

```
\((\w+):(\w+)(?:\s*\{.*?\})?\)
```

This regular expression captures nodes, including their aliases and types, such as `(p:Person)`.

For each node, the following steps are performed:

- **Existence check:** The method verifies whether the node type exists in the schema. For example, a node type `Person` in the query is validated against the list of nodes defined in the schema.
- **Error handling:** If the node type is not found, an error message is generated, such as:

  "Node type `Actor` is not defined in the schema."

4.1.2 Relationship validation

Relationships are extracted from the query using the pattern:

```
-\[:(\w+)\]->
```

This identifies relationships such as `-[:ACTED_IN]->`.

For each relationship, the following steps are performed:

- **Existence check:** The method ensures that the relationship type exists in the schema. For example, `ACTED_IN` is validated against the schema's list of relationships.
- **Error handling:** If the relationship type is invalid, an error message is generated, such as:

  "Relationship type `PLAYED_IN` is not defined in the schema."

4.1.3 Attribute validation

Attributes in the query are validated in two contexts: `WHERE` clauses and `RETURN` statements. The patterns used for extraction are:

```
WHERE (\w+)\.(\w+)
RETURN (\w+)\.(\w+)
```

The validation process involves:

- **WHERE clause attributes:** For each attribute in a `WHERE` clause, the method verifies whether the attribute exists for the specified node type in the schema.
- **RETURN clause attributes:** Similarly, attributes in the `RETURN` clause are validated against the schema.
- **Error handling:** If an attribute does not exist for a node type, an error message is generated. For example:

  "Attribute 'name' is not defined for node type 'Movie' in the schema."

4.1.4 Path-check exclusion

This implementation does not include a path-check functionality, which would validate whether the queried nodes and relationships form a valid traversable path in the knowledge graph. This exclusion was a deliberate design decision. Path-checking requires handling intricate dependencies between nodes and relationships to verify their logical connectivity within the graph. Developing and implementing such a feature would significantly increase the complexity of the schema compliance method, potentially diverting focus from the primary research objectives of this thesis.

Instead, the schema compliance method prioritizes verifying individual components (nodes, relationships, and attributes), which addresses most structural issues in queries. More on this can be read in 6.

4.2 Self-checking

While syntactic correctness and adherence to the database schema confirm that a query can be executed, these checks alone do not guarantee that the resulting query is meaningful. A way of improving the quality and correctness of automatically generated Cypher queries — both in terms of semantic and syntactic correctness — can be achieved through a self-checking strategy that leverages the LLM.

After generating a query from an LLM, inaccuracies may persist. These issues can include syntactic misalignments that are not caught by the schema compliance method, such as improper use of node or relationship names, as well as logical flaws in catching the user's intent in their request that will lead to semantically incorrect results. These should be filtered out by the schema compliance method, however, this is a handy double-check while also adding a semantic check. Queries may be syntactically correct while still failing to fulfill the user's intended information need, either by overlooking important conditions or introducing irrelevant constraints.

To try to mitigate these problems, the self-checking process employs an LLM in an evaluative capacity. By reintroducing the query, original user's request, and knowledge graph schema into the LLM, **this evaluation step utilizes the model's contextual and semantic understanding to determine whether the query accurately captures the user's objectives of their request and whether it follows the schema's requirements and rules.** The LLM is asked the following:

```
Is this query valid: {query}, given the following schema?
{schema}. And does it also capture the intent of this
prompt? {user's request prompt}. Only give a binary 'Yes'
or 'No' response. Something like: 'Yes' or 'No'.
```

This prompt forces the output to be a binary 'Yes' or 'No' answer (still a small filter is used to extract the first 'Yes' or 'No' from the output).

However, it is good to keep in mind that if the model is bad at the initial query generation topic, it is suspected that its evaluation skills are of the same bad quality. That is because a single LLM relies on the same underlying parameters and training data for both query generation and evaluation. In other words, if the model's internal representation of Cypher queries or domain concepts is incomplete or flawed, those same gaps will manifest during the evaluation process. The secondary evaluation essentially "inherits" the same misconceptions that caused the query's inaccuracies in the first place, making it less likely that incorrect logic or misalignments with user intent will be detected and corrected.

4.3 Re-interpretation

Another way to ensure semantic correctness is to convert the generated Cypher query back into plain English using an LLM, and then compare the meaning of that English explanation to the user's original request using an embedding model which translates both sentences to vector spaces.

The reasoning behind this approach is that if the pipeline truly encodes the user's request in the query, then translating the query back into natural language should yield a text that closely resembles the user's initial request. For instance, if the user says, 'Find all movies produced in 2020 that feature Tom Hanks as an actor', and the pipeline generates a Cypher query that matches Movie nodes and an Actor node named 'Tom Hanks' and filters by year 2020, a good LLM-based explanation of that Cypher query should highlight these same details. **By comparing this explanation to the user's original words in their request, we can assess how faithful the query is to the intended meaning.**

To perform the comparison, the pipeline makes use of an embedding model. The embedding model used can be read in 3.3. Semantic embeddings, derived from models like Sentence-BERT, allow both the original request and the query explanation to be converted into vector representations. Using this method the sentence 'It is sunny today' can be represented in a vector which basically is a list of a lot of numbers that are a condensed summary of the sentence's core ideas. It can look something like: $\begin{pmatrix} 0.84 \\ 0.42 \\ \vdots \\ 0.02 \end{pmatrix}$

Once both sentences are converted into numeric vectors, the model measures how 'close' these vectors are in the high-dimensional space. The approach the model takes is cosine similarity, which looks at the angle between two vectors. If the angle is small, it means the two sentences share a closer

meaning and will have a score that is closer to 1; if the angle is large, the sentences are more different in meaning and will have a score that is closer to 0. A negative score is also possible and means that the two sentences point in opposite directions. It indicates that the sentences are not just different, but their meanings may be closer to being opposed. Higher similarity scores indicate that the LLM-generated explanation is closely aligned with the user's request, suggesting that the underlying Cypher query is meaningful and faithful to the user's request.

It is theorized that an embedding-based comparison provides a quick, and most importantly, a quantifiable metric of similarity. In practice, this means the pipeline can systematically verify whether the LLM's explanation aligns with the user's original request—without relying solely on manual judgment—thereby increasing confidence in the pipeline's ability to generate meaningful Cypher queries.

5 Experiment results

Three schemas were selected to test the pipeline's performance across varying levels of complexity and size: the Movie schema (moderate complexity), the Twitter schema (high complexity), and the Airport schema (low complexity). **In total 166 different prompts were run.** There are 76 prompts from the Movie schema, 60 from the Twitter schema and 30 from the Airport schema.

Prompts requiring more than 30 retries were excluded from the evaluation method analysis due to time constraints in the experimentation phase. This cutoff led to the removal of 11 prompts (15%) for the Movie schema, 10 prompts (16%) for the Twitter schema, and 3 prompts (10%) for the Airport schema. A more in-depth analysis can be seen in figure 8, 9, 10 and 11. In those figures, 'difficulty' is referring to the prompt difficulty.

5.1 Cut off prompts

Figure 8 shows the amount of cut off prompts of all the schemas combined.

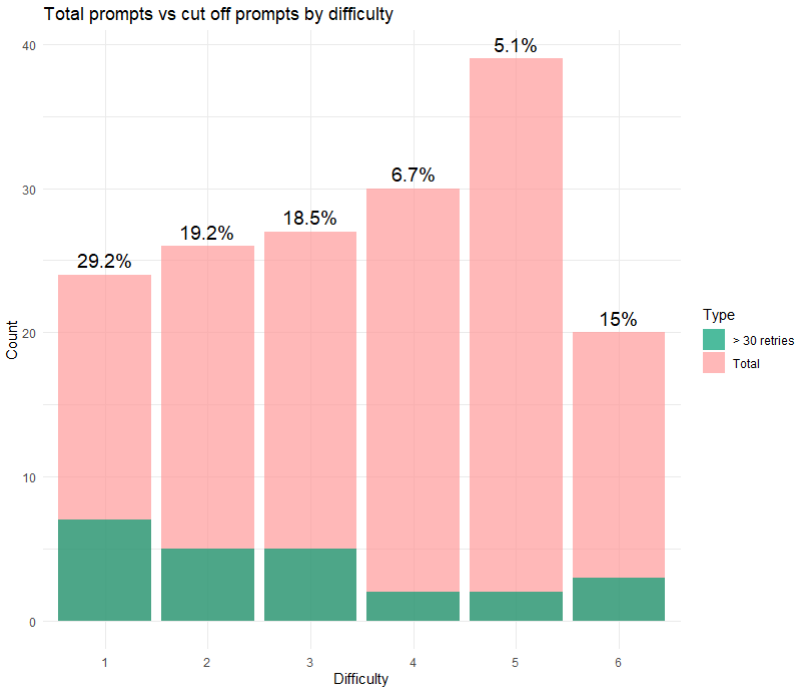

Figure 8: The amount of cut off prompts decrease as difficulty increases with an exception for difficulty 6. This could suggest the model is better at generating more complex valid queries than at simpler queries.

The cut off prompts can be seen per schema in figure 9, 10 and 11.

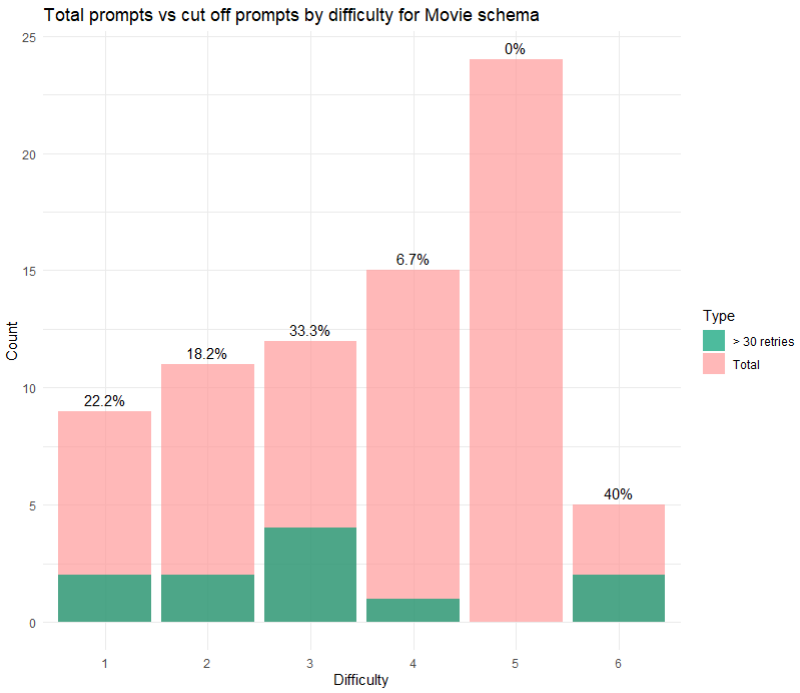

Figure 9: There is no clear pattern to be seen when compared to figure 8. A notable thing to see is, however, that no prompts have been cut off for difficulty 5, which contains the most amount of prompts.

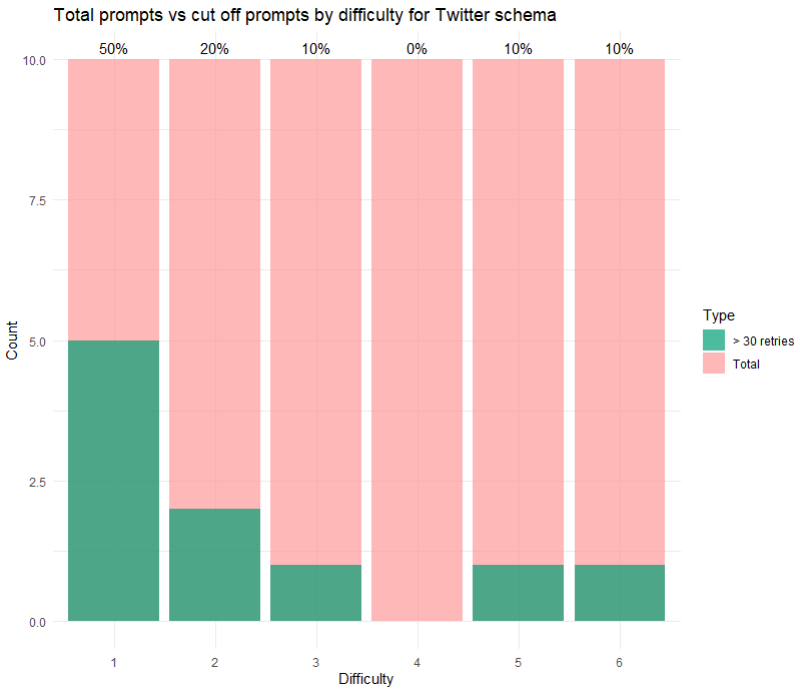

Figure 10: Also in this figure there is no clear pattern to be seen when compared to figure 8. The LLM had the most trouble generating queries for difficulty 1 prompts with the Twitter schema and is the cause for difficulty 1 being to highest cut off difficulty in figure 8. This could be because the information in those prompts was very limited and did not give the LLM enough context about the schema. Since this schema was the most complex the LLM could have needed more context about how the schema works, that it could extract from the prompt in higher difficulties.

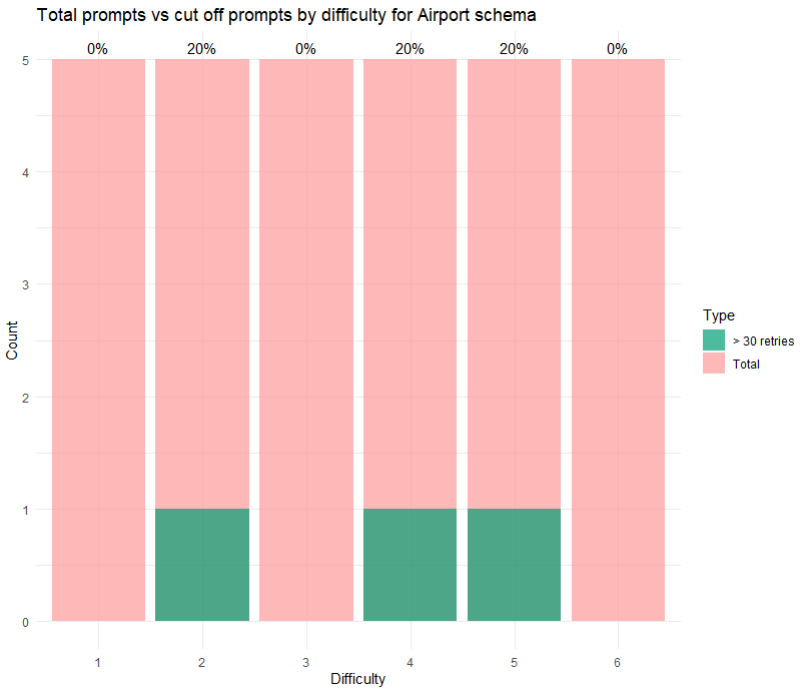

Figure 11: Again, in this figure there is no pattern to be seen when compared to figure 8. There is not one particular difficulty that shows notable behavior in this figure. This could be due to the fact that the schema itself was relatively simple and the LLM did not need to extract a lot of context from the prompt and had enough from the schema.

There does not seem to be a trend between the difficulty of the prompt and whether they have been cut off. It depends heavily upon the schema because no schema shares the same trend if there is even a trend to be seen. This could be due to the interplay between the schema's complexity and the model's ability to extract sufficient context from the prompt. Simpler schemas like the Airport schema, may allow the LLM to perform more consistently regardless of prompt difficulty, as the relationships and attributes in the schema are easier to interpret. Conversely, more complex schemas, like the Twitter schema, might require prompts that have more information in them for the model to generate valid queries, leading to a higher cutoff rate for the less complex prompts. This theory about the context is a hypothesis though and more testing is needed to be able to get an actual conclusion about these results. It will be discussed in section 6.

5.2 Evaluation methods

After the exclusion of prompts requiring more than 30 retries, the dataset included 65 usable prompts for the Movie schema, 50 for the Twitter schema, and 27 for the Airport schema.

To ensure a balanced analysis for the evaluation methods, three results per difficulty level were randomly selected from each schema (excluding the cut off prompts). **This sampling process resulted in a total of 54 prompts being analyzed across all schemas.** This selection strategy was implemented to provide a representative overview of the performance across varying levels of prompt complexity while maintaining manageability within the scope of the study.

Of the 54 prompts, 30 were manually evaluated to be correct while the other 24 were deemed incorrect. This results in the pipeline having an accuracy of 55.6% of generating syntactic and semantically correct queries in this sample.

All results should have been syntactically correct, as a query should only pass the schema compliance method if it meets the required syntactic standards. However, **some queries that were manually evaluated as incorrect still exhibited minor syntactic flaws.** In certain cases, some queries that the self-checking method deemed correct contained subtle errors.

5.2.1 Self-checking method

Table 2 presents the number of matches and mismatches between the self-checking method and the manual evaluation for each schema. The independent variable is the schema, where agreement between the self-checking method and manual evaluation is the dependent variable. Figure 12 provides a heat-map representation of the contingency table, highlighting the distribution of Type I errors (false positives) and Type II errors (false negatives).

Schema	Agreement	Count	Proportion
Movie	Match	12	0.222
Movie	Mismatch	6	0.111
Twitter	Match	11	0.204
Twitter	Mismatch	7	0.130
Airport	Match	13	0.241
Airport	Mismatch	5	0.093
Total	Match	36	0.666
Total	Mismatch	18	0.333

Table 2: Such a high error or mismatch rate (33%) is undesirable for this application and does not support the hypothesis, as it indicates that approximately one out of every three prompts receives an incorrect evaluation value from the self-checking method. Although the error rate is lower than 50%, this level of inaccuracy could undermine the reliability of the pipeline.

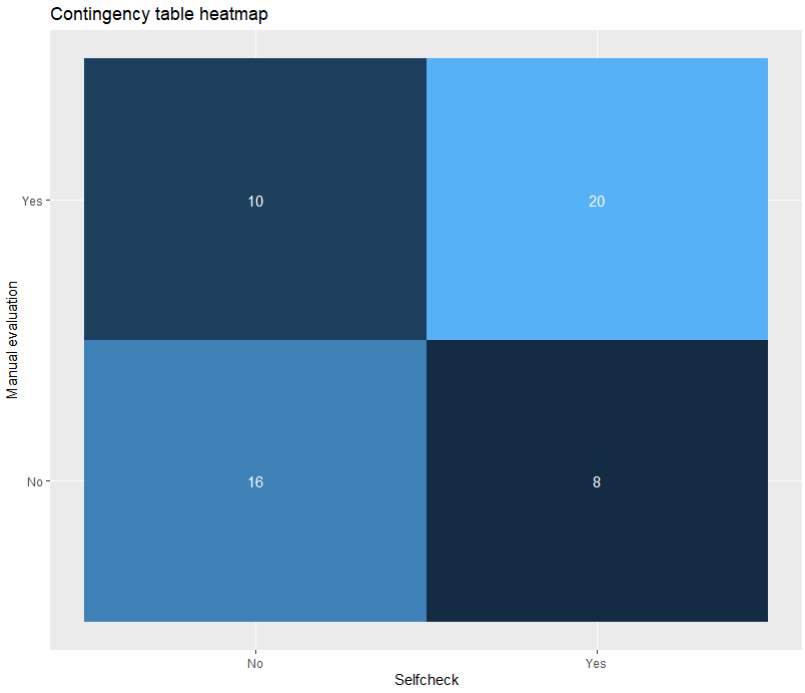

Figure 12: The implications of this error rate are significant: the 8 False Positives suggest that the self-checking method often overestimates positive cases, while the 10 False Negatives indicate that it also misses a substantial number of valid positive cases. This balance between Type I and Type II errors suggests that the method is not skewed toward one type of misclassification but is generally inconsistent in its predictions.

5.2.2 Re-interpretation method

Figure 13 shows a box-plot of the dependent variable similarity score and the independent variable being correctness. This is a visual representation of what table 3 shows. That contains a Welch Two Sample t-test on similarity score and correctness of the prompts.

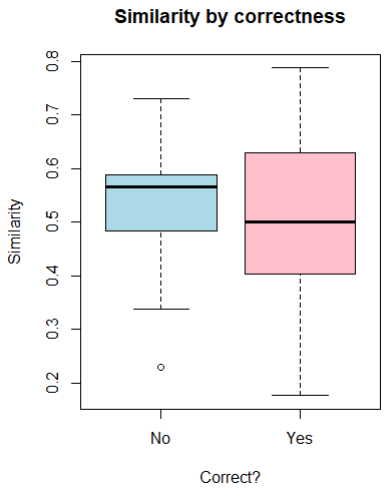

Figure 13: This figure reveals an unexpected pattern: in this sample of prompts, incorrect queries tend to have higher overall similarity scores compared to correct queries. The average score is higher and the spread is smaller. This outcome contradicts the intended goal of the method, which aimed to demonstrate that correct queries would show higher similarity scores with greater statistical significance.

Statistic	Value
t-value	0.920
df	49.795
p-value	0.362
95% confidence interval	-0.041 / 0.109
Effect size (Cohen's D)	0.240

Table 3: The degrees of freedom (df) are high due to the relatively large sample size (54), which typically increases the test's sensitivity to detect differences. However, there is no statistical significance between the similarity score and correctness, as indicated by a p-value of 0.362 (greater than the 0.05 threshold). This suggests that the reinterpreting evaluation method described in section 4.3 is neither effective nor particularly informative. The high p-value indicates a lack of strong evidence to support the hypothesis that high similarity scores are significantly related to correctness. The observed effect size is also relatively small, reinforcing the conclusion that the differences are minimal and unlikely to have any real relevance.

5.3 Prompt engineering

The results in table 4 highlight the varying degrees of success in aligning the LLM's output with the intended goal of retrieving information about the movie 'Cast Away'.

The results demonstrate that the LLM is generally effective at extracting the user's intent from a request, at least within the context of the Movie schema. Generally here means to not write a request with abolutely no context. It requires "relatively little" information to grasp at least part of the intended meaning. However, the LLM does not consistently capture the user's intent across all requests. To improve performance, **it is recommended to frame requests similarly to a well-structured Google query, or to how you would ask it to another person, ensuring they contain sufficient detail and context for the LLM to interpret accurately.** That is how "relatively little" information is needed.

#	**Prompt**	**Query / Result**
1	Find the movie 'Cast Away'	`MATCH (m:Movie {title:"Cast Away"})-[:ACTED_IN]->(p:Person) RETURN p.name, COLLECT(p.roles) AS roles;`
2	Cast Away	`MATCH (p:Person {name:"Tom Hanks"})-[:ACTED_IN]->(m1) WHERE size((p)-[:ACTED_IN]-(m)) >= 1 WITH p, collect(m) AS movies_acted_in UNWIND movies_acted_in AS m2 MATCH (d:Person)-[:DIRECTED]->(m3) WHERE d IN nodes(p)-[]->() RETURN DISTINCT m3;`
3	Movie Cast Away	`"MATCH p=(person:Person {name:""Tom Hanks""})-[:ACTED_IN]->(movie:Movie) <-[:REVIEWED]-(review) WHERE movie.title = 'Cast Away' AND review.rating = 8.1 RETURN person, movie, collect(review.summary) AS summaries"`
4	Find the movie Cast Away	`"MATCH p=(m:Movie {title:""Cast Away"", released:2000})-[:ACTED_IN]->(a:Person)-[r:ACTED_IN]-(p2) RETURN m.title AS title, a.name AS name, r.roles AS role;"`
5	Find the movie in which Tom Hanks played a guy that got stranded on an island	`"MATCH (p:Person {name:""Tom Hanks""})-[:ACTED_IN]->(m:Movie)<-[:HAS_TAGLINE]-(tagline) WHERE m.title CONTAINS ""castaway"" AND tagline.value = ""You can still make it home"" RETURN m;"`
6	Chuck Noland movie	`"MATCH (p:Person {name:'Chuck Noland'})-[:REVIEWED]->(m:Movie) RETURN m.title AS ""Title"", m.tagline AS ""Tagline"""`
7	WIIIILLLLSOOOON!	`MATCH (m:Movie {title:"The Shawshank Redemption"})-[:ACTED_IN]->(p:Person)-[r:REVIEWED]-(o) RETURN m, p, r, o;`

Table 4: Prompts 1 through 5 demonstrate that the LLM can recognize the intended focus on the movie, albeit with inconsistent syntactic correctness and alignment to the strict goal which is to retrieve the name of the movie. None of these queries meet both criteria simultaneously. More specifically, prompts 6 and 7 fail to link the user's input to Cast Away entirely. These prompts require substantial external knowledge about the movie itself, such as character names or iconic references, which may exceed the LLM's capacity for such domain-specific context.

6 Discussion and future work

6.1 Experimentation related

Retry limit selection: The decision to set a cutoff point of 30 retries for each prompt was driven by practical considerations, primarily time constraints. Certain prompts seemed problematic for the LLM, consistently failing to pass the schema compliance check. Waiting indefinitely for such prompts to succeed was not feasible, as it could have significantly delayed the experimentation process. However, it is worth noting that some prompts generated syntactically correct queries after 27 or 28 retries, suggesting that **a higher cutoff point might have allowed additional prompts to pass.** This, in turn, could have resulted in a more comprehensive dataset, potentially uncovering patterns in the LLM's failure modes and providing further insights into its behavior under different conditions. Future studies might explore the impact of varying the retry limit to determine whether higher thresholds yield significantly improved outcomes or reveal new insights about the LLM's performance.

Output variability and consistency: Another factor that could influence the results is the variability of the outputs on one prompt. LLMs often generate slightly different responses for the same input due to their inherent probabilistic nature. This means that even with identical prompts, the query generated in one attempt may differ from that in another. **These variations could affect the outcomes, especially when evaluating the semantic alignment or syntactic correctness of queries.** For instance, a prompt might eventually produce a syntactically correct query after several retries, while earlier attempts failed. This variability introduces an additional layer of randomness into the results, making it challenging to draw consistent conclusions. Future research could explore whether generating and averaging multiple outputs for the same prompt leads to more reliable evaluations.

6.2 Technical enhancements

Enhancing syntactic validation: Some queries passed the schema compliance check but still contained minor syntactic flaws as stated in chapter 5. Ensuring syntactic correctness without being able to execute the query is inherently challenging. Crafting regular expressions that account for every possible variation in query syntax is not only complex but often incomplete, as edge cases and unexpected patterns can easily slip through. A potential method to enhance syntactic evaluation, which has not been explored in this thesis but can be explored in future research, involves **executing the generated queries in a controlled test environment that mirrors the schema.** By running the queries and observing whether they produce

errors, this approach can guarantee that any query passing the test is syntactically correct. While more resource-intensive, such a method would provide an additional layer of reliability, ensuring that only executable queries are deemed valid. This method could also improve the performance of semantic evaluation methods, as it narrows their focus to queries that are guaranteed to be syntactically valid. By eliminating all invalid queries at an earlier stage, semantic evaluation methods would no longer need to account for issues caused by syntactic flaws. This could reduce noise in the evaluation process and lead to more accurate assessments of whether a query aligns with the user's intent, as the evaluations would solely target queries that are structurally correct and executable.

Incorporating negative examples: Another potential method for improving the initial query generation is to include examples of previous syntactically incorrect queries in the instructions provided to the LLM. By explicitly showing the model how not to construct a query, these **examples could serve as a guide for the LLM to avoid common mistakes.** This approach might help the LLM refine its understanding of the schema and the query language, potentially leading to more accurate outputs. However, the effectiveness of this technique has not been explored in this thesis and remains an open question for future research. Investigating whether such negative examples genuinely enhance query generation could provide valuable insights into improving the pipeline's performance.

Integrating path validation: Future research could explore integrating path validation as an enhancement to the schema compliance process, particularly in use cases where path correctness is critical to the accuracy of query results. While this study focused on validating individual components such as nodes, relationships, and attributes, the absence of path validation means that certain queries could pass compliance checks despite containing logically invalid paths. For example, a query could connect nodes using valid relationship types that do not form a meaningful traversal path according to the schema. Incorporating path validation would **ensure that the entire structure of a query adheres to the graph's schema, further increasing its reliability.** Future work could leverage graph traversal algorithms or dedicated libraries to implement efficient and scalable path validation mechanisms, potentially enhancing the robustness of the overall pipeline.

6.3 Model related

Choice of model: The results of the pipeline are heavily dependent on the model used. Since the evaluation methods show a lack of reliability in both the re-interpreting method producing semantic similarity score and the

self-checking method, it suggests that the limitations could not just lie in the methods themselves but also in the capabilities of the LLM employed. A less robust model may struggle with consistent schema awareness, nuanced intent understanding, or accurate evaluation of its own outputs. **It is possible that the model used is not good enough for this application.** These shortcomings highlight the importance of model selection and the potential need for employing domain-specific LLMs sufficiently trained on structured query languages. Without addressing these foundational topics, the methods may continue to yield suboptimal results. In the future the GPT-4o model (or any other by then available, more powerful and sufficiently trained model) could be used to see if results will improve.

Adjusting context size: Another potential area for exploration is the impact of the model's context size on the results. In this study, the context window provided to the LLM was selected to avoid any warnings about exceeding the model's capacity. However, the influence of context size on the quality of generated queries and their subsequent evaluation was not systematically tested. **It is possible that larger or smaller context windows could affect the model's ability to generate accurate and schema-compliant queries, especially for prompts with high complexity or schemas with extensive details.** Future research could experiment with varying the context size to determine its impact on performance. For instance, smaller context windows might hinder the model's ability to fully understand the schema, while larger ones might improve accuracy at the cost of increased computational requirements. Understanding these trade-offs could provide valuable insights into optimizing the model's performance for this specific application.

7 Conclusion

7.1 Answer to the research question

This thesis set out to investigate the following central question:

> *How can the quality and relevance of queries on knowledge graph generated by Large Language Models be automatically assessed and implemented, ensuring meaningful outputs aligned with the schema and user request without human supervision?*

To address this, a comprehensive pipeline was designed and tested on three different knowledge graph schemas of varying complexity (Movie, Twitter, and Airport). The pipeline (1) translates natural language requests into Cypher queries using an LLM, (2) enforces syntactic correctness through schema compliance checks, and (3) attempts to automatically evaluate semantic correctness via a self-checking and a re-interpretation method.

7.2 Summary of methods and key findings

- **Schema compliance:** Ensured that queries generated by the LLM are syntactically valid, matching the nodes, relationships, and attributes defined in each schema. However, still minor syntactic errors slipped through this step.
- **Self-checking:** Employed the same LLM to review its own output against the original prompt and schema for semantic alignment. Contrary to the hypothesis that it would achieve at least 80% agreement with manual evaluations, it only showed a **66% agreement rate**.
- **Re-interpretation (embedding-based similarity):** Converted the generated Cypher queries back into plain English and used an embedding model to measure the similarity between this "explanation" and the user's original request. Surprisingly, **higher similarity scores did *not* correlate ($p \not< 0.05$) with correct results**, failing to confirm the hypothesized strong correlation between semantic similarity score and true semantic correctness.

Overall, these findings show partial success. The pipeline automates much of the query generation process, but the evaluation methods—particularly semantic checks—need further refinement.

7.3 Unexpected findings

Two unexpected observations emerged during experimentation:

- **Mismatch in self-checking:** The self-checking approach often yielded incorrect judgments of correctness, suggesting that the same model generating a query may overlook—or even replicate—its own mistakes during evaluation.
- **Higher similarity score for incorrect queries:** Contrary to expectations, the re-interpretation method occasionally assigned higher similarity scores to *incorrect* queries than to correct ones, indicating that embedding-based methods can be influenced by other factors that do not necessarily reflect true semantic alignment.

7.4 Importance and implications

Despite these limitations, the proposed pipeline takes an important step toward democratizing access to knowledge graphs. By offloading the complexity of query formulation onto an LLM, non-experts can more readily explore complex data structures without deep expertise in query languages like Cypher. From a broader perspective:

- **Democratization of data:** With continued improvements, my pipeline could enable educators, researchers, and analysts to work with intricate knowledge graphs using natural language—lowering barriers to data-driven decision-making and analysis.
- **Scalable evaluation:** Automated methods, even when imperfect, have the potential to scale more readily than human evaluation, making them attractive for large or frequently updated data environments.
- **Future-ready tooling:** As more advanced or domain-specific LLMs become available, adopting those models could substantially boost both query generation and semantic evaluation, further enhancing the practicality of this pipeline.

In conclusion, while the current pipeline demonstrates that automated generation and evaluation of Cypher queries via LLMs is feasible, the evaluation mechanisms—particularly for semantic correctness—remain a challenge. Strengthening these components will be crucial for making LLM-driven interfaces robust enough to reliably serve non-experts, thus moving the field closer to the ultimate goal of truly democratizing knowledge graph access.

Bibliography

[1] G. Futia and A. Vetrò, "On the integration of knowledge graphs into deep learning models for a more comprehensible ai—three challenges for future research," *Information*, vol. 11, no. 2, p. 122, 2020.

[2] H. Abu-Rasheed, C. Weber, and M. Fathi, "Knowledge graphs as context sources for llm-based explanations of learning recommendations," *arXiv preprint arXiv:2403.03008*, 2024.

[3] T. A. van Schaik and B. Pugh, "A field guide to automatic evaluation of llm-generated summaries," in *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '24, (New York, NY, USA), p. 2832–2836, Association for Computing Machinery, 2024.

[4] M. Desmond, Z. Ashktorab, Q. Pan, C. Dugan, and J. M. Johnson, "Evalullm: Llm assisted evaluation of generative outputs," in *Companion Proceedings of the 29th International Conference on Intelligent User Interfaces*, IUI '24 Companion, (New York, NY, USA), p. 30–32, Association for Computing Machinery, 2024.

[5] A. Balcioglu, "Unveiling ai potential: Evaluating high-performance large language models in knowledge graph query interpretation," 2024.

[6] MetaAI, "Meta-llama-3-8b-instruct." `https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct`, 2024. Accessed: 2025-01-18.

[7] SentenceTransformers, "multi-qa-mpnet-base-cos-v1." `https://huggingface.co/sentence-transformers/multi-qa-mpnet-base-cos-v1`, 2024. Accessed: 2025-01-19.

A Knowledge graph schemas

A.1 Movies schema

```
nodes: [
{
  name: 'Movie',
  attributes: [
    {
      name: 'tagline',
      type: 'string',
    },
    {
      name: 'title',
      type: 'string',
    },
    {
      name: 'released',
      type: 'int',
    },
    {
      name: 'votes',
      type: 'int',
    },
  ],
},
{
  name: 'Person',
  attributes: [
    {
      name: 'born',
      type: 'int',
    },
    {
      name: 'name',
      type: 'string',
    },
  ],
},
],
edges: [
  {
    name: 'ACTED_IN',
    label: 'ACTED_IN',
```

```
  collection: 'ACTED_IN',
  from: 'Person',
  to: 'Movie',
  attributes: [
    {
      name: 'roles',
      type: 'string',
    },
  ],
},
{
  name: 'REVIEWED',
  label: 'REVIEWED',
  collection: 'REVIEWED',
  from: 'Person',
  to: 'Movie',
  attributes: [
    {
      name: 'summary',
      type: 'string',
    },
    {
      name: 'rating',
      type: 'int',
    },
  ],
},
{
  name: 'PRODUCED',
  label: 'PRODUCED',
  collection: 'PRODUCED',
  from: 'Person',
  to: 'Movie',
  attributes: [],
},
{
  name: 'WROTE',
  label: 'WROTE',
  collection: 'WROTE',
  from: 'Person',
  to: 'Movie',
  attributes: [],
},
{
```

```
    name: 'FOLLOWS',
    label: 'FOLLOWS',
    collection: 'FOLLOWS',
    from: 'Person',
    to: 'Person',
    attributes: [],
  },
  {
    name: 'DIRECTED',
    label: 'DIRECTED',
    collection: 'DIRECTED',
    from: 'Person',
    to: 'Movie',
    attributes: [],
  },
]
```

A.2 Twitter schema

```
nodes: [
{
  name: 'Me',
  attributes: [
    {
      name: 'screen_name',
      type: 'string',
    },
    {
      name: 'name',
      type: 'string',
    },
    {
      name: 'location',
      type: 'string',
    },
    {
      name: 'followers',
      type: 'int',
    },
    {
      name: 'following',
      type: 'int',
    },
    {
```

```
      name: 'url',
      type: 'string',
    },
    {
      name: 'profile_image_url',
      type: 'string',
    },
  ],
},
{
  name: 'Link',
  attributes: [
    {
      name: 'url',
      type: 'string',
    },
  ],
},
{
  name: 'Source',
  attributes: [
    {
      name: 'name',
      type: 'string',
    },
  ],
},
{
  name: 'Hashtag',
  attributes: [
    {
      name: 'name',
      type: 'string',
    },
  ],
},
{
  name: 'User',
  attributes: [
    {
      name: 'screen_name',
      type: 'string',
    },
    {
```

```
  name: 'name',
  type: 'string',
},
{
  name: 'location',
  type: 'string',
},
{
  name: 'followers',
  type: 'int',
},
{
  name: 'following',
  type: 'int',
},
{
  name: 'url',
  type: 'string',
},
{
  name: 'profile_image_url',
  type: 'string',
},
{
  name: 'screen_name',
  type: 'string',
},
{
  name: 'name',
  type: 'string',
},
{
  name: 'location',
  type: 'string',
},
{
  name: 'followers',
  type: 'int',
},
{
  name: 'following',
  type: 'int',
},
{
```

```
        name: 'statuses',
        type: 'int',
      },
      {
        name: 'url',
        type: 'string',
      },
      {
        name: 'profile_image_url',
        type: 'string',
      },
    ],
  },
  {
    name: 'Tweet',
    attributes: [
      {
        name: 'id',
        type: 'int',
      },
      {
        name: 'id_str',
        type: 'string',
      },
      {
        name: 'text',
        type: 'string',
      },
      {
        name: 'favorites',
        type: 'int',
      },
      {
        name: 'import_method',
        type: 'string',
      },
    ],
  },
],
edges: [
  {
    name: 'USING',
    label: 'USING',
    collection: 'USING',
```

```
  from: 'Tweet',
  to: 'Source',
  attributes: [],
},
{
  name: 'SIMILAR_TO',
  label: 'SIMILAR_TO',
  collection: 'SIMILAR_TO',
  from: 'User',
  to: 'User',
  attributes: [
    {
      name: 'score',
      type: 'float',
    },
  ],
},
{
  name: 'SIMILAR_TO',
  label: 'SIMILAR_TO',
  collection: 'SIMILAR_TO',
  from: 'User',
  to: 'Me',
  attributes: [
    {
      name: 'score',
      type: 'float',
    },
  ],
},
{
  name: 'AMPLIFIES',
  label: 'AMPLIFIES',
  collection: 'AMPLIFIES',
  from: 'Me',
  to: 'User',
  attributes: [],
},
{
  name: 'AMPLIFIES',
  label: 'AMPLIFIES',
  collection: 'AMPLIFIES',
  from: 'User',
  to: 'User',
```

```
  attributes: [],
},
{
  name: 'RT_MENTIONS',
  label: 'RT_MENTIONS',
  collection: 'RT_MENTIONS',
  from: 'Me',
  to: 'User',
  attributes: [],
},
{
  name: 'RT_MENTIONS',
  label: 'RT_MENTIONS',
  collection: 'RT_MENTIONS',
  from: 'User',
  to: 'User',
  attributes: [],
},
{
  name: 'FOLLOWS',
  label: 'FOLLOWS',
  collection: 'FOLLOWS',
  from: 'User',
  to: 'Me',
  attributes: [],
},
{
  name: 'FOLLOWS',
  label: 'FOLLOWS',
  collection: 'FOLLOWS',
  from: 'Me',
  to: 'User',
  attributes: [],
},
{
  name: 'FOLLOWS',
  label: 'FOLLOWS',
  collection: 'FOLLOWS',
  from: 'User',
  to: 'User',
  attributes: [],
},
{
  name: 'FOLLOWS',
```

```
  label: 'FOLLOWS',
  collection: 'FOLLOWS',
  from: 'Me',
  to: 'Me',
  attributes: [],
},
{
  name: 'INTERACTS_WITH',
  label: 'INTERACTS_WITH',
  collection: 'INTERACTS_WITH',
  from: 'User',
  to: 'User',
  attributes: [],
},
{
  name: 'INTERACTS_WITH',
  label: 'INTERACTS_WITH',
  collection: 'INTERACTS_WITH',
  from: 'Me',
  to: 'User',
  attributes: [],
},
{
  name: 'RETWEETS',
  label: 'RETWEETS',
  collection: 'RETWEETS',
  from: 'Tweet',
  to: 'Tweet',
  attributes: [],
},
{
  name: 'REPLY_TO',
  label: 'REPLY_TO',
  collection: 'REPLY_TO',
  from: 'Tweet',
  to: 'Tweet',
  attributes: [],
},
{
  name: 'CONTAINS',
  label: 'CONTAINS',
  collection: 'CONTAINS',
  from: 'Tweet',
  to: 'Link',
```

```
    attributes: [],
  },
  {
    name: 'MENTIONS',
    label: 'MENTIONS',
    collection: 'MENTIONS',
    from: 'Tweet',
    to: 'User',
    attributes: [],
  },
  {
    name: 'MENTIONS',
    label: 'MENTIONS',
    collection: 'MENTIONS',
    from: 'Tweet',
    to: 'Me',
    attributes: [],
  },
  {
    name: 'TAGS',
    label: 'TAGS',
    collection: 'TAGS',
    from: 'Tweet',
    to: 'Hashtag',
    attributes: [],
  },
  {
    name: 'POSTS',
    label: 'POSTS',
    collection: 'POSTS',
    from: 'User',
    to: 'Tweet',
    attributes: [],
  },
  {
    name: 'POSTS',
    label: 'POSTS',
    collection: 'POSTS',
    from: 'Me',
    to: 'Tweet',
    attributes: [],
  },
]
```

A.3 Airport schema

```
    nodes: [
    {
      name: 'Airports',
      attributes: [
        { name: 'City', type: 'string' },
        { name: 'Country', type: 'string' },
        { name: 'Lat', type: 'float' },
        { name: 'Long', type: 'float' },
        { name: 'Name', type: 'string' },
        { name: 'State', type: 'string' },
        { name: 'Vip', type: 'bool' },
      ],
    },
  ],
  edges: [
    {
      name: 'Flights',
      label: 'Flights',
      from: 'Airports',
      to: 'Airports',
      collection: 'Flights',
      attributes: [
        { name: 'ArrTime', type: 'int' },
        { name: 'ArrTimeUTC', type: 'string' },
        { name: 'Day', type: 'int' },
        { name: 'DayOfWeek', type: 'int' },
        { name: 'DepTime', type: 'int' },
        { name: 'DepTimeUTC', type: 'string' },
        { name: 'Distance', type: 'int' },
        { name: 'FlightNum', type: 'int' },
        { name: 'Month', type: 'int' },
        { name: 'TailNum', type: 'string' },
        { name: 'UniqueCarrier', type: 'string' },
        { name: 'Year', type: 'int' },
      ],
    },
  ]
```