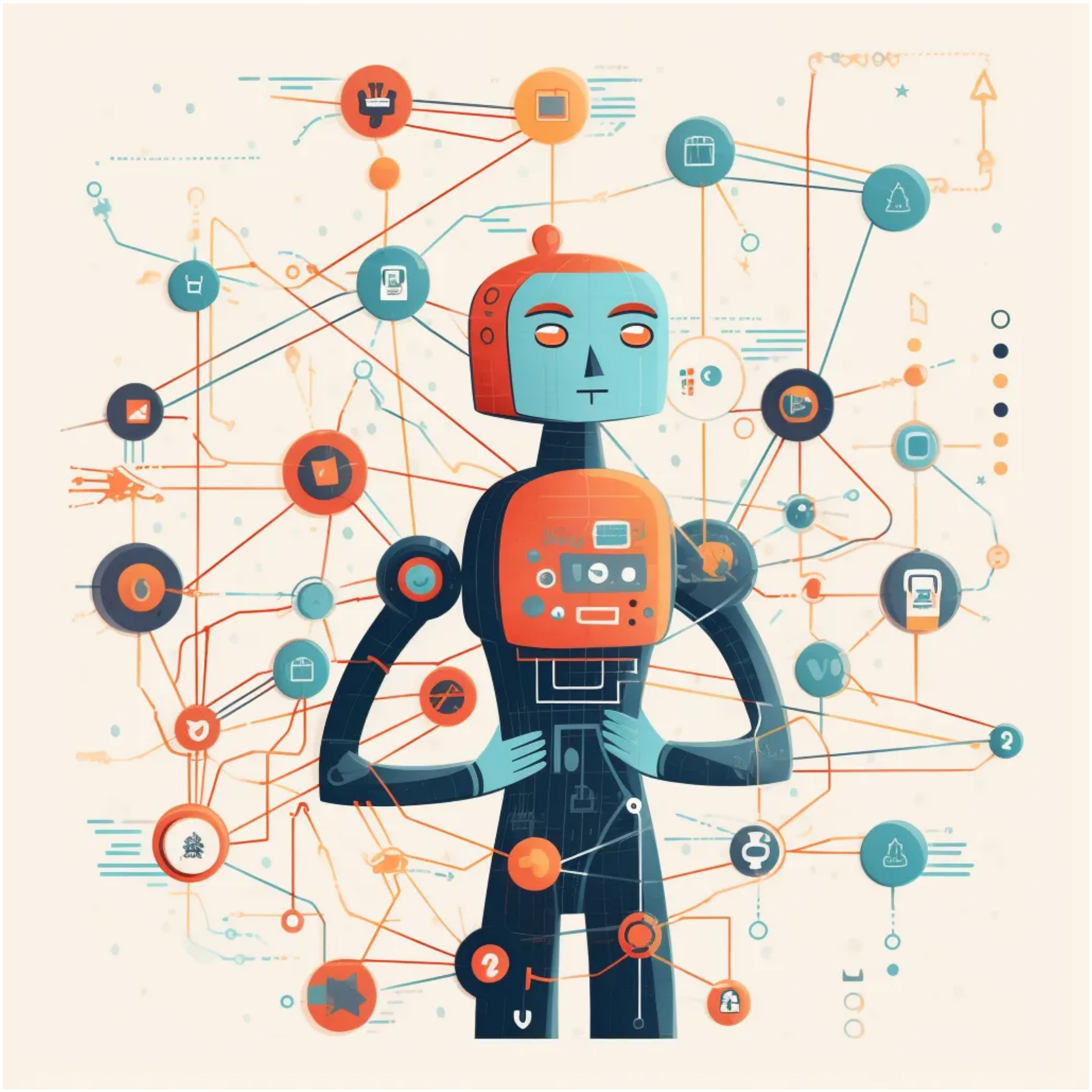
Search Medium

LangChain Cypher Search: Tips & Tricks

How to optimize prompts for Cypher statement generation to retrieve relevant information from Neo4j in your LLM applications

Tomaz Bratanic · Follow
Published in Neo4j Developer Blog
9 min read · Jun 2

Listen Share More



Knowledge-graph chatbot as imagined by Midjourney.

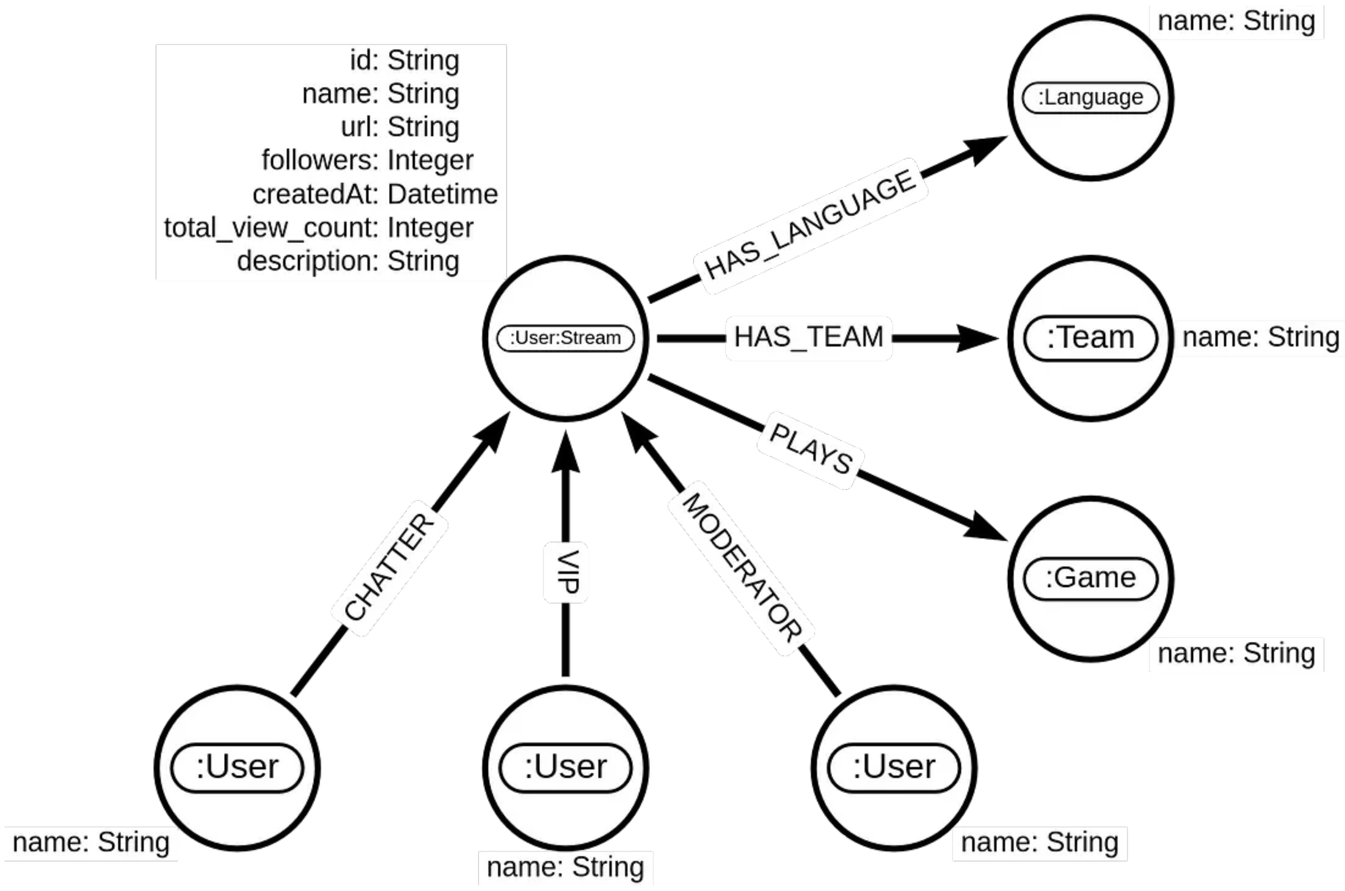
Last time, we looked at how to get started with Cypher Search in the LangChain library and why you would want to use knowledge graphs in your LLM applications. In this blog post, we will continue to explore various use cases for integrating knowledge graphs into LLM and LangChain applications. Along the way, you will learn how to improve prompts to produce better and more accurate Cypher statements.

Specifically, we will look at how to use the few-shot capabilities of LLMs by providing a couple of Cypher statement examples, which can be used to specify which Cypher statements the LLM should produce, what the results should look like, and more. Additionally, you will learn how you can integrate graph algorithms from the Neo4j Graph Data Science library into your LangChain applications.

All the code is available on GitHub.

Neo4j Environment Setup

In this blog post, we will be using the Twitch dataset that is available in Neo4j Sandbox.



Twitch dataset graph model. Image by the author.

The Twitch social network composes of users. A small percentage of those users broadcast their gameplay or activities through live streams. In the graph model, users who do live streams are tagged with a secondary label Stream. Additional information about which teams they belong to, which games they play on stream, and in which language they present their content is present. We also know how many followers they had at the moment of scraping, the all-time historical view count, and when they created their accounts. The most relevant information for network analysis is knowing which users engaged in the streamer's chat. You can distinguish if the user who chatted in the stream was a regular user (CHATTER relationship), a moderator of the stream (MODERATOR relationship), or a stream VIP.

The network information was scraped between the 7th and the 10th of May 2021. Therefore, the dataset has outdated information.

Improving LangChain Cypher Search

First, we have to set up the LangChain Cypher search.

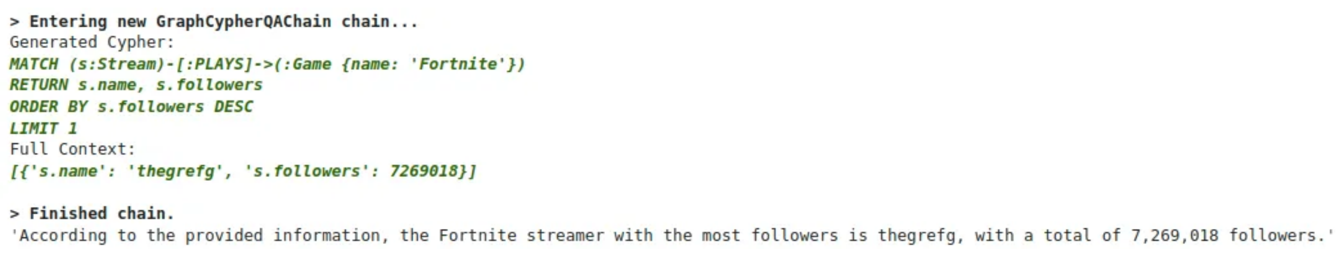
```
import os

from langchain.chat_models import ChatOpenAI
from langchain.chains import GraphCypherQAChain
from langchain.graphs import Neo4jGraph

os.environ['OPENAI_API_KEY'] = "OPENAI_API_KEY"

graph = Neo4jGraph(
    url="bolt://44.212.12.199:7687",
    username="neo4j",
    password="buoy-warehouse-subordinates"


chain = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0), graph=graph, verbose=True,
)
```

I really love how easy it is to setup the Cypher Search in the LangChain library. You only need to define the Neo4j and OpenAI credentials, and you are good to go. Under the hood, the `graph` objects inspect the graph schema model and pass it to the `GraphCypherQAChain` to construct accurate Cypher statements.

Let's begin with a simple question.

```
chain.run("""
Which fortnite streamer has the most followers?
""")
```

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)-[:PLAYS]->(:Game {name: 'Fortnite'})
RETURN s.name, s.followers
ORDER BY s.followers DESC
LIMIT 1
Full Context:
[{'s.name': 'thegrefg', 's.followers': 7269018}]

> Finished chain.
'According to the provided information, the Fortnite streamer with the most followers is thegrefg, with a total of 7,269,018 followers.'
```

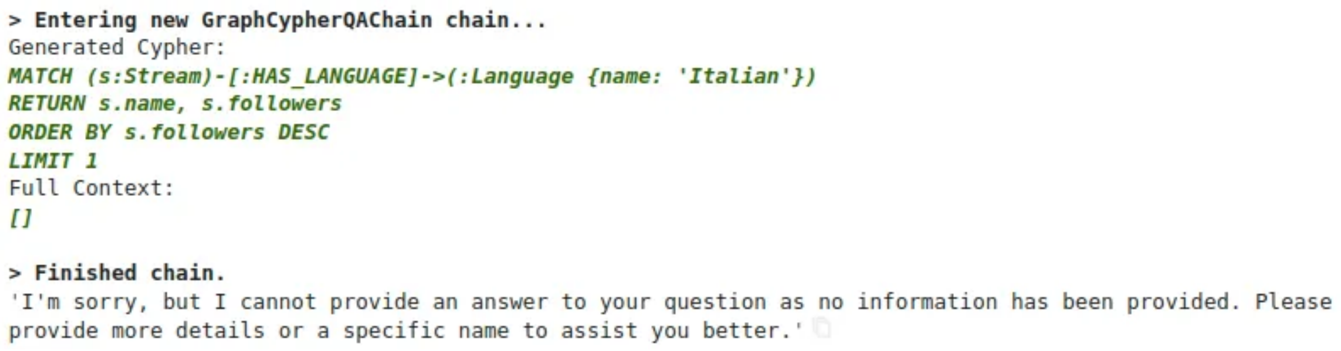
Generated answer. Image by the author.

The Cypher chain constructed a relevant Cypher statement, used it to retrieve information from Neo4j, and provided the answer in natural language form.

Now let's ask another question.

```
chain.run("""
Which italian streamer has the most followers?
""")
```

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)-[:HAS_LANGUAGE]->(:Language {name: 'Italian'})
RETURN s.name, s.followers
ORDER BY s.followers DESC
LIMIT 1
Full Context:
[]

> Finished chain.
'I'm sorry, but I cannot provide an answer to your question as no information has been provided. Please
provide more details or a specific name to assist you better.'
```

Generated answer. Image by the author.

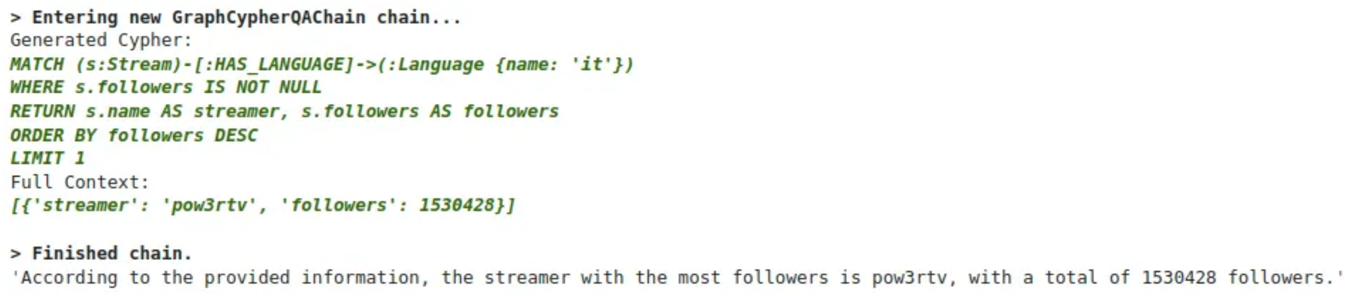
The generated Cypher statement looks valid, but unfortunately, we didn't get any results. The problem is that the language values are stored as two-character country codes, and the LLM is unaware of that. There are a few options we have to overcome this problem. First, we can utilize the few-shot capabilities of LLMs by providing examples of Cypher statements, which the model then imitates when generating Cypher statements. To add example Cypher statements in the prompt, we have to update the Cypher generating prompt. You can take a look at the default prompt used to generate Cypher statements to better understand the update we are going to do.

```
from langchain.prompts.prompt import PromptTemplate


CYPHER_GENERATION_TEMPLATE = """
Task:Generate Cypher statement to query a graph database.
Instructions:
Use only the provided relationship types and properties in the schema.
Do not use any other relationship types or properties that are not provided.
Schema:
{schema}
Cypher examples:
# How many streamers are from Norway?
MATCH (s:Stream)-[:HAS_LANGUAGE]->(:Language {{name: 'no'}})
RETURN count(s) AS streamers

Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than for you to co
Do not include any text except the generated Cypher statement.

The question is:
{question}"""
CYPHER_GENERATION_PROMPT = PromptTemplate(
    input_variables=["schema", "question"], template=CYPHER_GENERATION_TEMPLATE
)
```

If you compare the new Cypher generating prompt to the default one, you can observe we only added the Cypher examples section. We added an example where the model could observe that the language values are given as two-character country codes. Now we can test the improved Cypher chain to answer the question about the most followed Italian streamers.

```
chain_language_example = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0), graph=graph, verbose=True,
    cypher_prompt=CYPHER_GENERATION_PROMPT
)

chain_language_example.run("""
Which italian streamer has the most followers?
""")
```

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)-[:HAS_LANGUAGE]->(:Language {name: 'it'})
WHERE s.followers IS NOT NULL
RETURN s.name AS streamer, s.followers AS followers
ORDER BY followers DESC
LIMIT 1
Full Context:
[{'streamer': 'pow3rtv', 'followers': 1530428}]

> Finished chain.
'According to the provided information, the streamer with the most followers is pow3rtv, with a total of 1530428 followers.'
```

Generated answer. Image by the author.

The model is now aware that the languages are given as two-character country codes and can now accurately answer questions that use the language information.

Another way we could solve this problem is to give a few example values of each property when defining the graph schema information. The solution would be generic and probably quite useful. Perhaps time to add another PR to the LangChain library :)

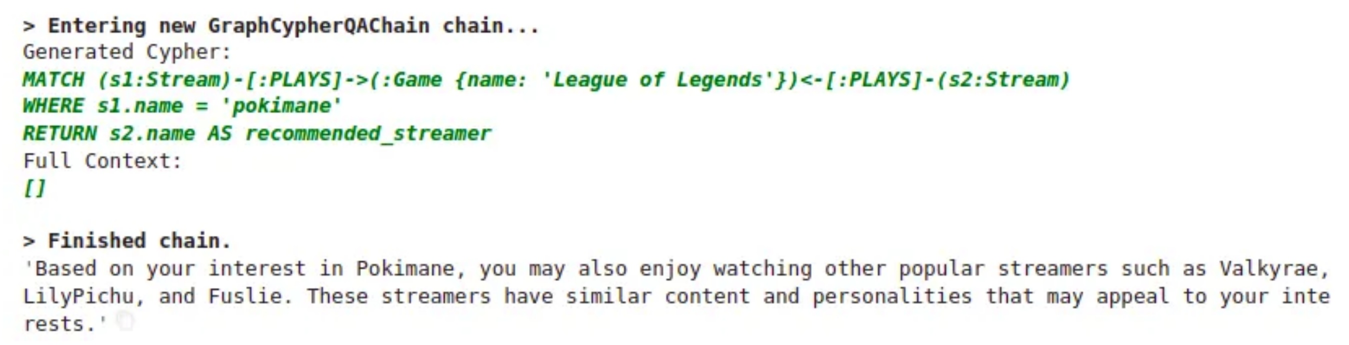
Using graph algorithms to answer questions

In the previous blog post, we looked at how integrating graph databases into LLM applications can answer questions like how entities are connected by finding the shortest or other paths between them. Today we will look at other use cases where graph databases can be used in LLM applications that other databases struggle with, specifically how we can use graph algorithms like PageRank to provide relevant answers. For example, we can use personalized PageRank to provide recommendations to an end user at query time.

Take a look at the following example:

```
chain_language_example.run("""
Which streamers should also I watch if I like pokimane?
""")
```

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s1:Stream)-[:PLAYS]->(:Game {name: 'League of Legends'})<-[:PLAYS]-(s2:Stream)
WHERE s1.name = 'pokimane'
RETURN s2.name AS recommended_streamer
Full Context:
[]

> Finished chain.
'Based on your interest in Pokimane, you may also enjoy watching other popular streamers such as Valkyrae,
LilyPichu, and Fuslie. These streamers have similar content and personalities that may appeal to your inte
rests.'
```

Generated answer. Image by the author.

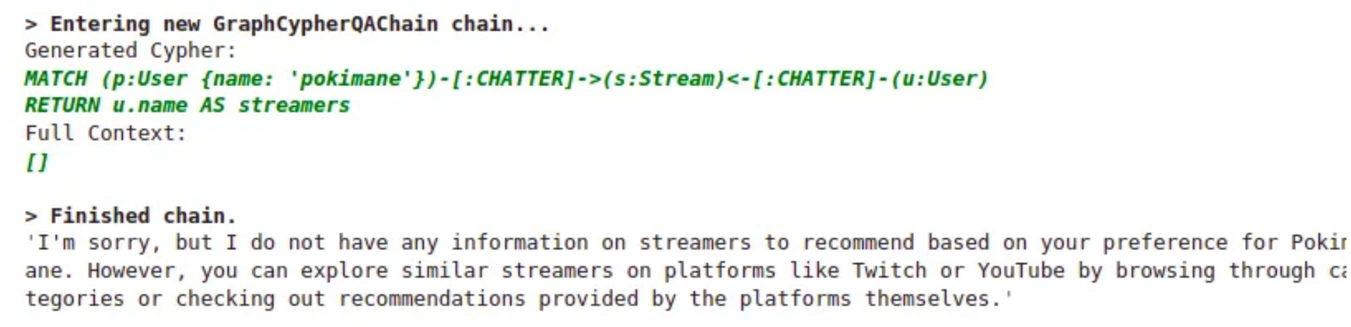
Interestingly, every time we rerun this question, the model will generate a different Cypher statement. However, one thing is consistent. For some reason, every time the League of Legends is somehow included in the query.

A bit more worrying fact is that the LLM model provided recommendations even though it wasn't provided with any suggestions in the prompt context. It's known that **gpt-3.5-turbo** sometimes doesn't follow the rules, especially if you do not repeat them more than once.

GPT-3.5-turbo being itself.

Repeating the instruction three times can help gpt-3.5-turbo solve this problem. However, by repeating instructions, you are increasing the token count and, consequently, the cost of Cypher generation. Therefore, it would take some prompt engineering to get the best results using the lowest count of tokens.

On the other hand, **GPT-4** is far better at following instructions.



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (p:User {name: 'pokimane'})-[:CHATTER]->(s:Stream)<-[:CHATTER]-(u:User)
RETURN u.name AS streamers
Full Context:
[]

> Finished chain.
'I'm sorry, but I do not have any information on streamers to recommend based on your preference for Pokir
ane. However, you can explore similar streamers on platforms like Twitch or YouTube by browsing through ca
tegories or checking out recommendations provided by the platforms themselves.'
```

GPT-4 doesn't add any information that is not available in the context. Image by the author.

GPT-4 didn't add any information from its internal knowledge. However, its generated Cypher statement was still relatively bad. Again, we can solve this problem by providing Cypher examples in the LLM prompt.

As mentioned, we will use Personalized PageRank to provide stream recommendations. But first, we need to project the in-memory graph and run the Node Similarity algorithm to prepare the graph to be able to give recommendations. Look at my previous blog post to learn more about how graph algorithms can be used to analyze the Twitch network.

```
# Project in-memory graph
graph.query("""
CALL gds.graph.project('shared-audience',
  ['User', 'Stream'],
  {CHATTER: {orientation:'REVERSE'}})
""")

# Run node similarity algorithm
graph.query("""
CALL gds.nodeSimilarity.mutate('shared-audience',
 {similarityMetric: 'Jaccard',similarityCutoff:0.05, topK:10, sudo:true,
     mutateProperty:'score', mutateRelationshipType:'SHARED_AUDIENCE'})
""")
```

The node similarity algorithm will take about 30 seconds to complete as the database has almost five million users. The Cypher statement to provide recommendations using Personalized PageRank is the following:

```
MATCH (s:Stream)
WHERE s.name = "kimdoe"
WITH collect(s) AS sourceNodes
CALL gds.pageRank.stream("shared-audience",
  {sourceNodes:sourceNodes, relationshipTypes:['SHARED_AUDIENCE'],
    nodeLabels:['Stream']})
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
WHERE NOT node in sourceNodes
RETURN node.name AS streamer, score
ORDER BY score DESC LIMIT 3
```

The OpenAI LLMs could be better at using the Graph Data Science library as their knowledge cutoff is September 2021, and version 2 of the Graph Data Science library was released in April 2022. Therefore, we need to provide another example in the prompt to show the LLM show to use Personalized PageRank to give recommendations.

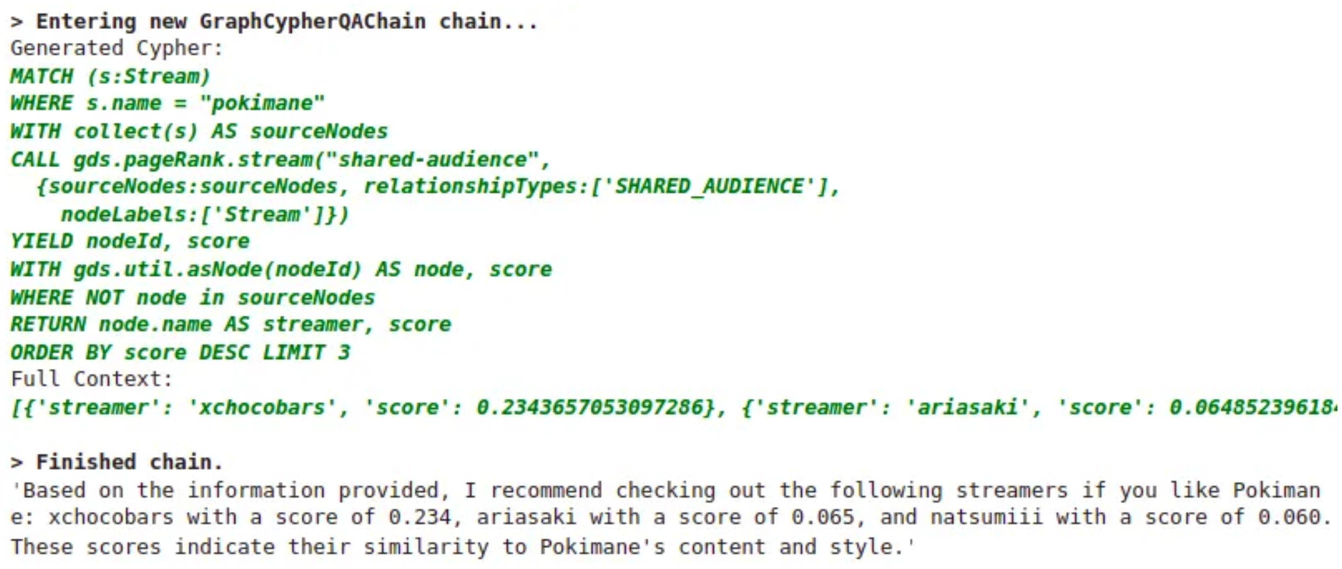
```
CYPHER_RECOMMENDATION_TEMPLATE = """Task:Generate Cypher statement to query a g
Instructions:
Use only the provided relationship types and properties in the schema.
Do not use any other relationship types or properties that are not provided.
Schema:
{schema}
Cypher examples:
# How many streamers are from Norway?
MATCH (s:Stream)-[:HAS_LANGUAGE]->(:Language {{name: 'no'}})
RETURN count(s) AS streamers
# Which streamers do you recommend if I like kimdoe?
MATCH (s:Stream)
WHERE s.name = "kimdoe"
WITH collect(s) AS sourceNodes
CALL gds.pageRank.stream("shared-audience",
  {sourceNodes:sourceNodes, relationshipTypes:['SHARED_AUDIENCE'],
    nodeLabels:['Stream']})
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
WHERE NOT node in sourceNodes
RETURN node.name AS streamer, score
```

```
ORDER BY score DESC LIMIT 3

Note: Do not include any explanations or apologies in your responses.
Do not respond to any questions that might ask anything else than for you to co
Do not include any text except the generated Cypher statement.

The question is:
{question}"""
CYPHER_RECOMMENDATION_PROMPT = PromptTemplate(
    input_variables=["schema", "question"], template=CYPHER_RECOMMENDATION_TEMP
)
```

We can now test the Personalized PageRank recommendations.

```
chain_recommendation_example = GraphCypherQAChain.from_llm(
    ChatOpenAI(temperature=0, model_name='gpt-4'), graph=graph, verbose=True,
    cypher_prompt=CYPHER_RECOMMENDATION_PROMPT,
)

chain_recommendation_example.run("""
Which streamers do you recommend if I like pokimane?
""")
```

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)
WHERE s.name = "pokimane"
WITH collect(s) AS sourceNodes
CALL gds.pageRank.stream("shared-audience",
  {sourceNodes:sourceNodes, relationshipTypes:['SHARED_AUDIENCE'],
    nodeLabels:['Stream']})
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
WHERE NOT node in sourceNodes
RETURN node.name AS streamer, score
ORDER BY score DESC LIMIT 3
Full Context:
[{'streamer': 'xchocobars', 'score': 0.2343657053097286}, {'streamer': 'ariasaki', 'score': 0.06485239618

> Finished chain.
'Based on the information provided, I recommend checking out the following streamers if you like Pokiman
e: xchocobars with a score of 0.234, ariasaki with a score of 0.065, and natsumiii with a score of 0.060.
These scores indicate their similarity to Pokimane's content and style.'
```

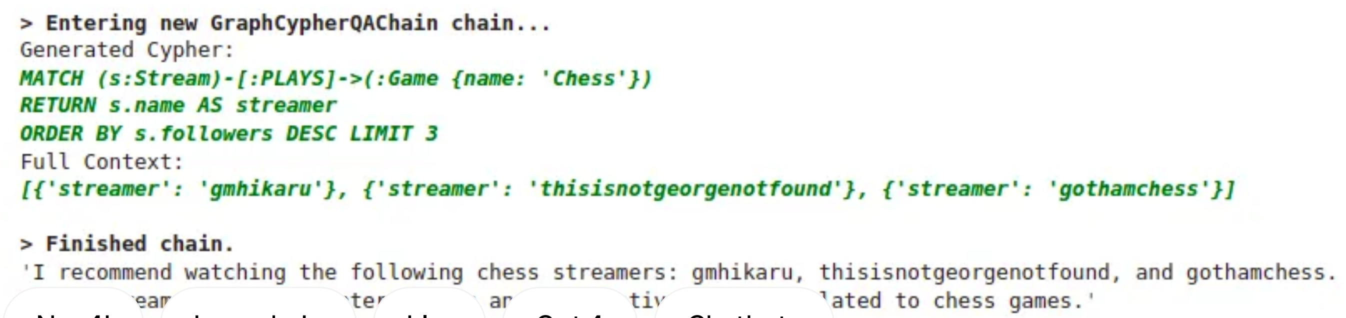
Generated answer. Image by the author.

Unfortunately, here, we have to use the GPT-4 model as the gpt-3.5-turbo is stubborn and doesn't want to imitate the complex Personalized PageRank example.

We can also test if the GPT-4 model will decide to generalize the Personalized PageRank recommendation in other use cases.

```
chain_recommendation_example.run("""
Which streamers do you recommend to watch if I like Chess games?
""")
```

Results



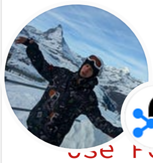
```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)-[:PLAYS]->(:Game {name: 'Chess'})
RETURN s.name AS streamer
ORDER BY s.followers DESC LIMIT 3
Full Context:
[{'streamer': 'gmhikaru'}, {'streamer': 'thisisnotgeorgenotfound'}, {'streamer': 'gothamchess'}]

> Finished chain.
'I recommend watching the following chess streamers: gmhikaru, thisisnotgeorgenotfound, and gothamchess.
        eam            ter       ar        tiv            lated to chess games.'
```

Neo4j Langchain Llm Gpt 4 Chatbots

Generated answer. Image by the author.

The LLM decided to take a more straightforward route to provide recommendations and simply returned the three chess players with the highest follower count. We can't really blame it for choosing this option.

However, LLMs are quite good at listening to hints:



```
recommendation_example.run("""
treamers do you recommend to watch if I like Chess games?
se Personalized PageRank to provide recommendations.
Do not exclude sourceNodes in the answer
```

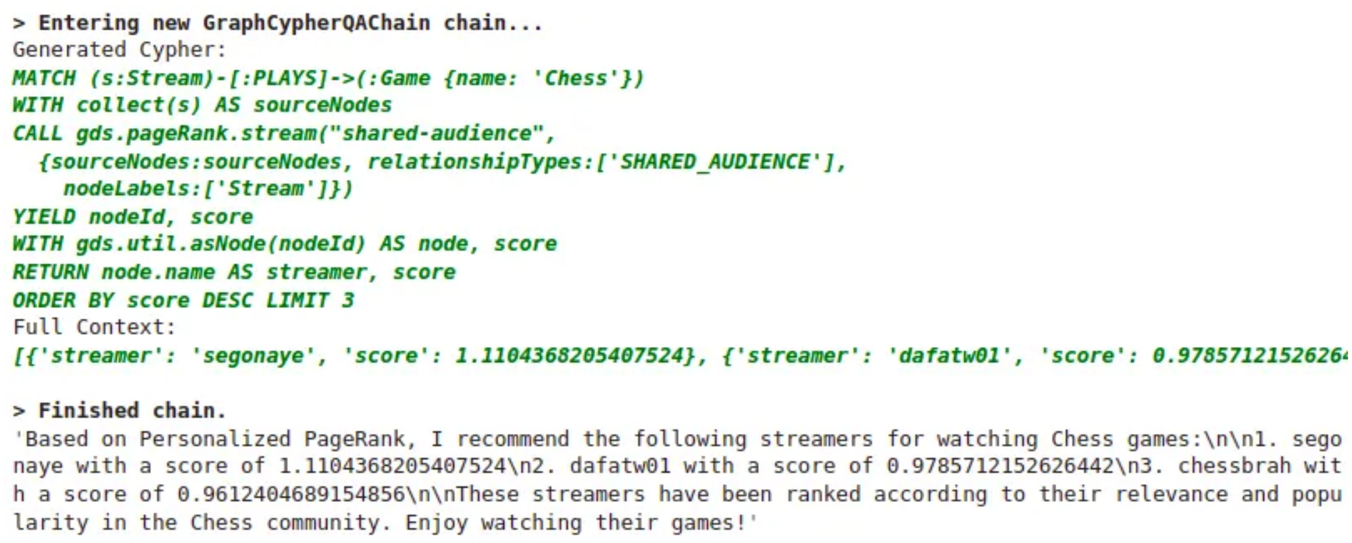
Follow

Written by Tomaz Bratanic

3.5K Followers · Writer for Neo4j Developer Blog

Data explorer. Turn everything into a graph. Author of Graph algorithms for Data Science at Manning publication. http://mng.bz/GGVN

Results



```
> Entering new GraphCypherQAChain chain...
Generated Cypher:
MATCH (s:Stream)-[:PLAYS]->(:Game {name: 'Chess'})
WITH collect(s) AS sourceNodes
CALL gds.pageRank.stream("shared-audience",
  {sourceNodes:sourceNodes, relationshipTypes:['SHARED_AUDIENCE'],
    nodeLabels:['Stream']})
YIELD nodeId, score
WITH gds.util.asNode(nodeId) AS node, score
RETURN node.name AS streamer, score
ORDER BY score DESC LIMIT 3
Full Context:
[{'streamer': 'segonaye', 'score': 1.1104368205407524}, {'streamer': 'dafatw01', 'score': 0.97857121526264

> Finished chain.
'Based on Personalized PageRank, I recommend the following streamers for watching Chess games:\n\n1. sego
naye with a score of 1.1104368205407524\n2. dafatw01 with a score of 0.9785712152626442\n3. chessbrah wit
h a score of 0.9612404689154856\n\nThese streamers have been ranked according to their relevance and popu
larity in the Chess community. Enjoy watching their games!'
```

main opportunity to improve the Cypher generation accuracy is to use the few-shot capabilities of LLMs, offering Cypher statement examples that dictate the type of statements an LLM should produce. Sometimes, the LLM model doesn't correctly guess the property values, while other times, it doesn't provide the Cypher statements we would like it to generate. Additionally, we have looked at how we can use graph algorithms like Personalized PageRank in LLM applications to provide better and more relevant answers.

As always, the code is available on GitHub.

Tomaz Bratanic in Towards Data Science

LangChain has added Cypher Search

With the LangChain library, you can conveniently generate Cypher queries, enabling an efficient retrieval of information from Neo4j.

8 min read · May 25

142 7