Improving LLM-based KGQA for multi-hop Question Answering with implicit reasoning in few-shot examples

Mili Shah, Joyce Cahoon, Mirco Milletari, Jing Tian
Fotis Psallidas, Andreas Mueller, Nick Litombe
Microsoft
milishah, jcahoon, mimillet, jingtian, fopsalli, amueller, nicklitombe@microsoft.com

Abstract

Large language models (LLMs) have shown remarkable capabilities in generating natural language texts for various tasks. However, using LLMs for question answering on knowledge graphs still remains a challenge, especially for questions requiring multi-hop reasoning. In this paper, we present a novel planned query guidance approach that improves large language model (LLM) performance in multi-hop question answering on knowledge graphs (KGQA). We do this by designing few-shot examples that implicitly demonstrate a systematic reasoning methodology to answer multi-hop questions. We evaluate our approach for two graph query languages, Cypher and SPARQL, and show that the queries generated using our strategy outperform the queries generated using a baseline LLM and typical few-shot examples by up to 24.66% and 7.7% in execution match accuracy for the MetaQA and the Spider benchmarks respectively. We also conduct an ablation study to analyze the incremental effects of the different techniques of designing few-shot examples. Our results suggest that our approach enables the LLM to effectively leverage the few-shot examples to generate queries for multi-hop KGQA.

1 Introduction

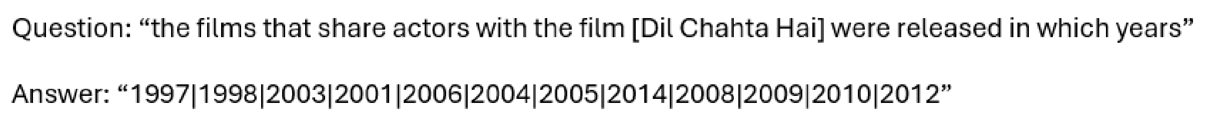
Question answering on knowledge graphs (KGQA) is a challenging task that requires understanding the natural language query, mapping it to the KG schema, and generating a graph query that can retrieve the correct answer from the KG. We focus on two graph query languages in this work, namely Cypher[1], a well-known graph query language developed by Neo4j[2], and SPARQL, a popular language for querying RDF[3] databases. In this study, we focus on the task of answering a question from a knowledge graph by generating Cypher and SPARQL queries to query the knowledge graph.

Large language models (LLMs), such as GPT-4 (OpenAI et al., 2024), have shown remarkable capabilities in generating natural language texts for various tasks. Recent studies have explored the capability of LLMs in Cypher generation (Guo et al., 2023; Jiang et al., 2023b; An et al., 2023) as well as SPARQL generation (Jiang et al., 2023a; Li et al., 2023; Gu and Su, 2022; Ye et al., 2021). However, using LLMs for multi-hop KGQA still remains a challenge, as they need to generate queries that can capture the multi-hop reasoning logic. Furthermore, models are limited by the availability of labeled data for KGQA, which is costly and time-consuming to obtain.

Therefore, it is desirable to leverage the few-shot learning ability of LLMs, which allows them to adapt to new tasks with only a few examples, and design effective few-shot examples that can guide the LLM to generate more accurate queries for multi-hop KGQA. The utilization of few-shot learning in LLMs has shown promise in various domains to address the limitations of data scarcity and improve model generalization. Several studies demonstrated the value of few-shot learning in various domains for improving the performance of LLMs (Shirafuji et al., 2023; Huang et al., 2024; Ahmed and Devanbu, 2023). However, to the best of our knowledge, the influence of few-shot examples design in KGQA, particularly for generating Cypher and SPARQL queries, has not been extensively studied.

Existing KGQA models like TransferNet(Shi et al., 2021), which excels in multi-hop reasoning over relation graphs, UniKGQA(Jiang et al., 2022), known for its unified retrieval and reasoning framework, and NuTrea(Choi et al., 2023), which leverages neural tree search for context-rich embeddings, outperform our proposed LLM-based

[1] https://neo4j.com/docs/cypher-manual/current/introduction/
[2] https://neo4j.com/
[3] Resource Description Framework


Question: “the films that share actors with the film [Dil Chahta Hai] were released in which years”

Answer: “1997|1998|2003|2001|2006|2004|2005|2014|2008|2009|2010|2012”

Figure 1: An example of a 3-hop question-answer pair in MetaQA.

approach, achieving up to 100% in the Hits@1 metric. However, these methods incur higher complexity and cost, as they require extensive training on specific knowledge graphs. In contrast, simple LLM-based methods can achieve competitive performance with a well-designed few-shot example set, bypassing the need for exhaustive training or customization. This efficiency makes the study of few-shot example design for LLM-based KGQA a crucial research area, promising swift adaptability and innovation in question-answering systems.

In this paper, we propose a novel approach to improve the performance of LLM-based Cypher and SPARQL generation for multi-hop KGQA. We do this by designing few-shot examples that implicitly demonstrate a systematic reasoning methodology to answer multi-hop questions. This guides the LLM to follow a similar reasoning process for new questions, without explicitly specifying the steps. We hypothesize that such few-shot examples can enhance the LLM's understanding of the question, the KG schema, and the syntax of the graph query language, enabling it to generate more accurate queries for multi-hop KGQA.

We evaluate our approach on two popular benchmark datasets, MetaQA (Zhang et al., 2018) and Spider (Kosten et al., 2023), both of which feature natural language questions across various levels of difficulty for multi-hop querying. We start by conducting an ablation study to analyze the effects of different components of our few-shot examples design on Cypher. We then show how our methodology transfers to SPARQL. Our results demonstrate that this strategy can enhance execution match accuracy over that of conventional methods used in few-shot examples.

2 Methodology

This work focuses on the methodology of crafting few-shot examples for improved performance of LLMs for the task of Cypher and SPARQL generation for KGQA. For this task, a few-shot example is composed of a natural language question, accompanied by an expected response of a Cypher or SPARQL query that can be run on an associated KG to answer the natural language question.

We propose a method for designing Cypher and SPARQL queries for few-shot examples that clearly demonstrates to the LLM the reasoning required to answer multi-hop questions. Techniques like chain-of-thought prompting (Wei et al., 2022) use textual explanations to teach step-by-step reasoning to LLMs. Our methodology employs a code style that implicitly shows how to take each hop step-by-step. Figure 2 is an example of a Cypher query written in such a style to be used as a few-shot example.

Contrast the query in Figure 2 with a typical or conventional style[4] used by developers to write Cypher queries in Figure 3. Certain prevalent practices characterize this conventional style of crafting such graph queries. These include the utilization of succinct and non-descriptive variable names, the consolidation of all traversal hops into a single chain in a single MATCH clause, and the immediate specification of string literals for entity matching within the variable declaration itself in the MATCH clause (e.g., {name: “Dil Chahta Hai”}).

Our proposed approach outlines practical methods for crafting few-shot examples to generate graph queries, like Cypher and SPARQL:

1. **Structured Traversal Clarity**: Each hop should be articulated on a separate line to mirror the logical sequence of traversals, strictly adhering to the correct order of entities and relationships encountered. This makes the traversal reasoning clear and easy to follow. This approach enhances the clarity of traversal reasoning, ensuring that each step is both transparent and sequentially accurate.

2. **Logical Continuity in Chaining**: Maintain an unbroken logical chain where the endpoint

[4] https://neo4j.com/docs/cypher-manual/current/styleguide/, https://neo4j.com/docs/cypher-manual/current/queries/basic/#find-connected-nodes, https://gist.github.com/wjgilmore/8ba5f31ef1435dc04c52, https://gist.github.com/wjgilmore/8ba5f31ef1435dc04c52

```
MATCH (dilMovie:`movie`)-[:starred_actors]->(actor:`actor`)
MATCH (actor:`actor`)<-[:starred_actors]-(otherMovie:`movie`)
MATCH (otherMovie:`movie`)-[:release_year]->(year:`year`)
WHERE toLower(dilMovie.name)='dil chahta hai'
AND dilMovie <> otherMovie
RETURN year LIMIT 200
```

Figure 2: A sample of a Cypher query used in a few-shot example designed using our approach. Implicit reasoning is demonstrated by writing each hop line-by-line, with an easy-to-understand code style following the correct chain of hops, and separating reasoning of hops from the constraints into the WHERE clause. The natural language question corresponding to this Cypher query from the MetaQA dataset is "the films that share actors with the film Dil Chahta Hai were released in which years".

```
MATCH (yr:`year`)<-[:release_year]-(m:`movie`)-[:starred_actors]->(:`actor`)<-
[:starred_actors]-(m2:`movie` {name: 'Dil Chahta Hai'})
WHERE m <> m2
RETURN yr LIMIT 200
```

Figure 3: A sample of a Cypher query written in a commonly used style. The natural language question corresponding to this Cypher query from the MetaQA dataset is "the films that share actors with the film Dil Chahta Hai were released in which years".

of one hop is the starting point for the next, ensuring a coherent flow of entities throughout the query. Ensuring a coherent progression of entities throughout the query facilitates the LLM's ability to mirror the thought process when identifying subsequent steps.

3. **Distinct Separation of Logic**: In the case of Cypher, employ MATCH clauses exclusively for hops, while isolating all constraints, such as string literals for entity matching, within WHERE clauses to promote clarity; and in the case of SPARQL, utilize WHERE clauses for hops and separate constraints within the FILTER clause. This approach delineates the decision-making process for selecting hops from other constraints, thus sharpening the focus on the hop selection mechanism.

4. **Descriptive Variable Naming**: Adopt variable names that are both illustrative and consistent, reflecting the entity type and any applicable constraints, such as "dilMovie" to denote a 'movie' entity constrained by the title "Dil Chahta Hai". This approach enhances the traversal's logical coherence as well as aids the LLM in retaining the constraints for inclusion in the WHERE clause.

5. **Examples with increasing complexity**: Present multiple examples that escalate in complexity, such as starting with a simple 1-hop query and advancing through to more complex 2-hop and 3-hop queries, to reinforce the learning of the reasoning pattern.

6. **Consistency**: Ensure that the structure and presentation of all few-shot examples remain uniform, facilitating easier pattern recognition and learning.

3 Experimental setup

3.1 Cypher

3.1.1 Dataset

We evaluate our approach on a widely used benchmark for multi-hop KGQA, MetaQA (Zhang et al., 2018). MetaQA comprises of a movie knowledge graph with 43k entities and 9 relationship types, along with question-answer pairs. The dataset contains 161 1-hop question templates (31% of total question templates), 210 2-hop question templates (40% of total question templates), and 150 3-hop question templates (29% of total question templates). The corresponding answers are a list of entities from the KG. Figure 1 shows an example of a 3-hop question-answer pair.

3.1.2 Baselines

We compare our proposed approach with an LLM-based Cypher generation module[5] developed by

[5]https://python.langchain.com/docs/use_cases/graph/integrations/graph_cypher_qa

Question Type	Our proposed approach	LangChain's Cypher generation module	Examples with conventional style
3-hop (150 questions)	**97.33%**	67.33%	72.67%
2-hop (210 questions)	**100%**	89.52%	93.33%
1-hop (161 questions)	**92.54%**	88.81%	91.30%

Table 1: KGQA results for the MetaQA dataset comparing our approach (with systematic few-shot examples implicitly demonstrating reasoning), with two baselines, the first being the Cypher generation module available in LangChain, and the second being an approach where Cypher queries in few-shot examples are written in a typical fashion. The metric shown is execution match accuracy.

Variation of few-shot example design	Execution match accuracy
Conventionally written examples (baseline)	72.67%
Only one example written conventionally	83.33%
Non-descriptive variable names	87.33%
All hops in one line	94.67%
Only one example written with our design	95.33%
Chain direction not maintained	95.33%
Examples written with our design	**97.33%**

Table 2: Ablation experiments on 3-hop questions of the MetaQA dataset. Appendix D provides the few-shot examples used for each of these experiments.

Neo4j and made available in LangChain. This is a commonly used module for the task of KGQA.

We also compare against a second baseline of few-shot examples with Cypher queries written in a typical or conventional fashion. An example of a Cypher query written in such a style is shown in Figure 3. Section 2 details some features that characterize this conventional style. This baseline enables us to determine the influence of the design of few-shot examples.

3.1.3 Query Generation and Post-Processing pipeline

Our experiments employ GPT-4 (OpenAI et al., 2024) as the LLM across all methods under examination to generate Cypher queries given natural language questions.

We run these generated Cypher queries on MetaQA KG hosted in Neo4j.

A Cypher query corrector module[6] is incorporated as a post-processing step to rectify common errors in the directionality of relationships within the Cypher queries. For instance, it corrects MATCH (dilMovie:‘movie‘)<–[:starred_actors]–(actor:‘actor‘) to MATCH (dilMovie:‘movie ‘)–[:starred_actors]–>(actor:‘actor‘). To ensure consistency and fairness in our comparative analysis, the Cypher query corrector module is applied across all experimental conditions, encompassing the proposed approach, the baselines, and all the ablation studies.

[6]https://api.python.langchain.com/en/latest/chains/langchain.chains.graph_qa.cypher_utils.CypherQueryCorrector.html

3.1.4 Prompt

For both the proposed approach and the baseline with typically written Cypher queries, we specify three few-shot examples. The ablation studies employ variations of few-shot examples. All other prompt components, namely the instructions and the graph schema, remain consistent across these two methods as well as ablation studies. The complete prompt utilized for our proposed approach, including all the few-shot examples, is detailed in Appendix A. The few-shot examples employed for the baseline featuring typically written examples are enumerated in Appendix C.

For the baseline that relies on LangChain's Cypher generation module, we use the default prompt generated by the module. Notably, it does not involve any few-shot examples. The complete prompt is attached in Appendix B.

3.2 SPARQL

3.2.1 Dataset

In order to evaluate how well this style of few-shot expression generalizes and transfers to other graph

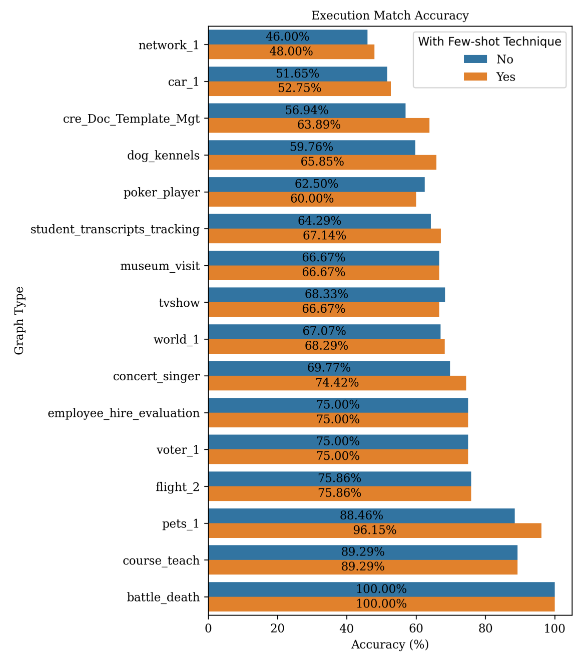


Figure 4: Execution match accuracy of our methodology for generating SPARQL queries on the Spider benchmark.

query languages, we tested its applicability on the SPARQL version of the Spider benchmark (Kosten et al., 2023). This version of Spider is useful as it already includes few-shot examples for training, precluding the need for us to handcraft examples as done for our Cypher module. We leverage a static prompt as attached in Appendix E to convert these few-shot examples into the style as outlined in Section 2.

3.2.2 Baselines

We compare the performance of the few-shots re-expressed in the format discussed against the performance without these few-shot examples. The prompt follows that as utilized for Cypher generation, except the graph schema is provided in RDF format. Similar to the Cypher pipelines above, we also include a corrector module as a post-processing step across these strategies to ensure that the SPARQL generated is syntactically valid.

3.3 Evaluation metric

The success of the generated queries is determined by the accuracy of the output, specifically, whether the entities in the generated answers precisely align with those anticipated in the expected answers. Execution match is reported in terms of the number of samples meeting this criterion.

4 Results

4.1 On Cypher generation

We observe in Table 1 that our approach outperforms both the LLM-based KGQA system in LangChain and the baseline of few-shot examples in terms of exact match accuracy across all hop levels. The increase in performance is especially pronounced in 3-hop questions, supporting our hypothesis that our methodology is able to effectively demonstrate to the LLM the reasoning required to answer complex multi-hop questions.

Notably, our proposed approach shows better performance in 3-hop and 2-hop questions over 1-hop questions. Manual examination revealed that most of the failures in 1-hop questions can be attributed to confusion between selecting the correct entity-type to traverse between "imdbvotes" and "imdbrating" for questions like "how famous of a film was [Pumping Iron]" or "what do people think of [Beau Travail] ".

The results in Table 2 show that including three examples instead of one in a typically written style leads to regression in performance, and thus demonstrates the importance of well-designed examples. Other ablation experiments show that other features of example design in our approach like using descriptive variable names, writing the hops in order of traversal, etc. contribute positively to performance. Few-shot examples used for each ablation experiment are listed in Appendix D.

4.2 On SPARQL generation

Figure 4 highlights the performance of our few-shot design against the baseline across 16 different knowledge graphs from the SPARQL version of the Spider benchmark (Kosten et al., 2023). There is a modest increase in execution match accuracy for SPARQL, with the most significant improvement—-a 7.7% lift-—observed in the subset of questions related to the pets_1 graph. There are six graphs where our methodology shows no improvement in execution match accuracy, and two graphs, `tvshow` and `poker_player`, where it leads to regressions. This outcome primarily stems from the use of few-shot examples that do not quite match the query complexity for the question set associated with those graphs, as classified by the original benchmark's measure of query hardness, which includes queries with 10+ hops. Conducting a paired difference t-test on these results yields a test statistic of 2.33 with a corresponding p-value of 0.03, indicating that the minor lift provided by our methodology is statistically significant.

5 Discussion

Our findings demonstrate the effectiveness of our proposed approach in improving the performance of LLM-based KGQA systems, particularly in addressing the challenge of multi-hop reasoning. By designing few-shot examples that implicitly demonstrate systematic reasoning to guide LLMs in generating Cypher and SPARQL queries, we have shown enhancements in accuracy, thereby highlighting the potential of this methodology for advancing the field of KGQA. Future research directions include testing our proposed approach on knowledge graphs with increasingly complex schemas, addressing challenges such as accurate attribute selection, aggregations and function usage in these graph query languages, and assessing the efficacy of using this few-shot example design in more graph languages and code generation tasks. Additionally, there is potential to develop techniques that automatically generate few-shot examples for a broad range of LLMs, streamlining the creation process and enhancing adaptability across various domains.

References

Toufique Ahmed and Premkumar Devanbu. 2023. Few-shot training llms for project-specific code-summarization. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, ASE '22, New York, NY, USA. Association for Computing Machinery.

Yuan An, Jane Greenberg, Alex Kalinowski, Xintong Zhao, Xiaohua Hu, Fernando J. Uribe-Romo, Kyle Langlois, Jacob Furst, and Diego A. Gómez-Gualdrón. 2023. Knowledge graph question answering for materials science (kgqa4mat): Developing natural language interface for metal-organic frameworks knowledge graph (mof-kg). *Preprint*, arXiv:2309.11361.

Hyeong Kyu Choi, Seunghun Lee, Jaewon Chu, and Hyunwoo J Kim. 2023. Nutrea: Neural tree search for context-guided multi-hop kgqa. In *Advances in Neural Information Processing Systems*, volume 36, pages 35954–35965. Curran Associates, Inc.

Yu Gu and Yu Su. 2022. Arcaneqa: Dynamic program induction and contextualized encoding for knowledge base question answering. *arXiv preprint arXiv:2204.08109*.

Jiayan Guo, Lun Du, Hengyu Liu, Mengyu Zhou, Xinyi He, and Shi Han. 2023. Gpt4graph: Can large language models understand graph structured data ? an empirical evaluation and benchmarking. *Preprint*, arXiv:2305.15066.

Xijie Huang, Li Lyna Zhang, Kwang-Ting Cheng, Fan Yang, and Mao Yang. 2024. Fewer is more: Boosting llm reasoning with reinforced context pruning. *Preprint*, arXiv:2312.08901.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Wayne Xin Zhao, and Ji-Rong Wen. 2023a. Structgpt: A general framework for large language model to reason over structured data. *arXiv preprint arXiv:2305.09645*.

Jinhao Jiang, Kun Zhou, Zican Dong, Keming Ye, Xin Zhao, and Ji-Rong Wen. 2023b. StructGPT: A general framework for large language model to reason over structured data. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 9237–9251, Singapore. Association for Computational Linguistics.

Jinhao Jiang, Kun Zhou, Wayne Xin Zhao, and Ji-Rong Wen. 2022. Unikgqa: Unified retrieval and reasoning for solving multi-hop question answering over knowledge graph. *arXiv preprint arXiv:2212.00959*.

Catherine Kosten, Philippe Cudré-Mauroux, and Kurt Stockinger. 2023. Spider4sparql: A complex benchmark for evaluating knowledge graph question answering systems. In *2023 IEEE International Conference on Big Data (BigData)*, pages 5272–5281. IEEE.

Tianle Li, Xueguang Ma, Alex Zhuang, Yu Gu, Yu Su, and Wenhu Chen. 2023. Few-shot in-context learning for knowledge base question answering. *arXiv preprint arXiv:2305.01750*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott

Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Jiaxin Shi, Shulin Cao, Lei Hou, Juanzi Li, and Hanwang Zhang. 2021. TransferNet: An effective and transparent framework for multi-hop question answering over relation graph. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 4149–4158, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

A. Shirafuji, Y. Oda, J. Suzuki, M. Morishita, and Y. Watanobe. 2023. Refactoring programs using large language models with few-shot examples. In *2023 30th Asia-Pacific Software Engineering Conference (APSEC)*, pages 151–160, Los Alamitos, CA, USA. IEEE Computer Society.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed H. Chi, Quoc V. Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *NeurIPS*.

Xi Ye, Semih Yavuz, Kazuma Hashimoto, Yingbo Zhou, and Caiming Xiong. 2021. Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. *arXiv preprint arXiv:2109.08678*.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J. Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *Proceedings of the Thirty-Second AAAI Conference on Artificial Intelligence and Thirtieth Innovative Applications of Artificial Intelligence Conference and Eighth AAAI Symposium on Educational Advances in Artificial Intelligence*, AAAI'18/IAAI'18/EAAI'18. AAAI Press.

A An example of a full prompt employing our design

```
[
  {
    "role": "system",
    "content": "You are a Cypher query
    generation engine for a Neo4j graph with the
     schema described below. You will create
    Cypher queries that helps users navigate the
    graph.

---
Node types:
– writer
– movie
– actor
– director
– tag
– year
– language
– genre
```

– imdbrating
– imdbvotes

Relations schema:
movie –> directed_by –> actor
movie –> directed_by –> director
movie –> directed_by –> writer
movie –> has_genre –> genre
movie –> has_imdb_rating –> imdbrating
movie –> has_imdb_votes –> imdbvotes
movie –> has_tags –> tag
movie –> in_language –> language
movie –> release_year –> year
movie –> starred_actors –> actor
movie –> starred_actors –> director
movie –> starred_actors –> writer
movie –> written_by –> actor
movie –> written_by –> director
movie –> written_by –> writer
–––"
  },
  {
    "role": "user",
    "content": "Write a Cypher query for: \"what films can be described by occupation\""
  },
  {
    "role": "assistant",
    "content": "‘‘‘
MATCH (movie:‘movie‘)–[:has_tags]–>(occupationTag:‘tag‘)
WHERE toLower(occupationTag.name)='occupation'
RETURN movie LIMIT 200
‘‘‘"
  },
  {
    "role": "user",
    "content": "Write a Cypher query for: \"which person wrote the films directed by Yuriy Norshteyn\""
  },
  {
    "role": "assistant",
    "content": "‘‘‘
MATCH (yuriyDirector:‘director‘)<–[:directed_by]–(movie:‘movie‘)
MATCH (movie:‘movie‘)–[:written_by]–>(writer:‘writer‘)
WHERE toLower(yuriyDirector.name)='yuriy norshteyn'
RETURN writer LIMIT 200
‘‘‘"
  },
  {
    "role": "user",
    "content": "Write a Cypher query for: \"the films that share actors with the film Dil Chahta Hai were released in which years\""
  },
  {
    "role": "assistant",
    "content": "‘‘‘
MATCH (dilMovie:‘movie‘)–[:starred_actors]–>(actor:‘actor‘)
MATCH (actor:‘actor‘)<–[:starred_actors]–(otherMovie:‘movie‘)
MATCH (otherMovie:‘movie‘)–[:release_year]–>(year:‘year‘)
WHERE toLower(dilMovie.name)='dil chahta hai'
AND dilMovie <> otherMovie
RETURN year LIMIT 200
‘‘‘"
  },
  {
    "role": "user",
    "content": "Write a Cypher query for: \"what types are the films starred by actors in The Gypsy Moths\""
  }
]

B An example of a full prompt used in the baseline using LangChain's Cypher generation module

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:
Node properties are the following:
writer {name: STRING, node_id: STRING}, movie {name: STRING, node_id: STRING}, actor {name: STRING, node_id: STRING}, director {name: STRING, node_id: STRING },tag {name: STRING, node_id: STRING}, year {name: STRING, node_id: STRING}, language {name: STRING, node_id: STRING},genre {name: STRING, node_id:

```
    STRING},imdbrating {name: STRING,
    node_id: STRING},imdbvotes {name:
    STRING, node_id: STRING}
Relationship properties are the following:
directed_by {source: STRING},written_by {
    source: STRING},starred_actors {source:
    STRING},release_year {source: STRING},
    in_language {source: STRING},has_tags {
    source: STRING},has_genre {source:
    STRING},has_imdb_votes {source:
    STRING},has_imdb_rating {source:
    STRING}
The relationships are the following:
(:movie)–[:has_tags]–>(:tag),(:movie)–[:
    directed_by]–>(:writer),(:movie)–[:
    directed_by]–>(:actor),(:movie)–[:
    directed_by]–>(:director),(:movie)–[:
    written_by]–>(:writer),(:movie)–[:
    written_by]–>(:actor),(:movie)–[:written_by
    ]–>(:director),(:movie)–[:in_language]–>(:
    language),(:movie)–[:release_year]–>(:year)
    ,(:movie)–[:has_genre]–>(:genre),(:movie)
    –[:starred_actors]–>(:actor),(:movie)–[:
    starred_actors]–>(:director),(:movie)–[:
    starred_actors]–>(:writer),(:movie)–[:
    has_imdb_rating]–>(:imdbrating),(:movie)
    –[:has_imdb_votes]–>(:imdbvotes)
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 construct a
    Cypher statement.
Do not include any text except the generated
    Cypher statement.
The question is:
the movies that share actors with the movie [
    Indiana Jones and the Last Crusade] were in
    which languages
```

C Few-shot examples provided to LLM for the baseline of typically written examples

```
Question: what films can be described by
    occupation
Cypher query: ‘‘‘
MATCH (s:‘movie‘)–[r:has_tags]–>(o:‘tag‘)
WHERE toLower(o.name)=’occupation’
    RETURN s LIMIT 200
‘‘‘
```

```
Question: which person wrote the films directed
    by Yuriy Norshteyn
Cypher query: ‘‘‘
MATCH (d:‘director‘ {name:’Yuriy Norshteyn’})
    <–[:directed_by]–(m:‘movie‘)–[:written_by
    ]–>(w:‘writer‘) RETURN w LIMIT 200
‘‘‘
```

```
Question: the films that share actors with the film
    Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (yr:‘year‘)<–[:release_year]–(m:‘movie
    ‘)–[:starred_actors]–>(:‘actor‘)<–[:
    starred_actors]–(m2:‘movie‘ {name: ’Dil
    Chahta Hai’})
WHERE m <> m2
RETURN yr LIMIT 200
‘‘‘
```

D Few-shot examples provided to LLM for ablation experiments

D.1 Ablation experiment "One typical example only"

```
Question: the films that share actors with the film
    Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (yr:‘year‘)<–[:release_year]–(m:‘movie
    ‘)–[:starred_actors]–>(:‘actor‘)<–[:
    starred_actors]–(m2:‘movie‘ {name: ’Dil
    Chahta Hai’})
WHERE m <> m2
RETURN yr LIMIT 200
‘‘‘
```

D.2 Ablation experiment "Non-descriptive variable names"

```
Question: what films can be described by
    occupation
Cypher query: ‘‘‘
MATCH (m:‘movie‘)–[:has_tags]–>(t:‘tag‘)
WHERE toLower(t.name)=’occupation’
RETURN m LIMIT 200
‘‘‘
```

```
Question: which person wrote the films directed
    by Yuriy Norshteyn
Cypher query: ‘‘‘
MATCH (d:‘director‘)<–[:directed_by]–(m:‘
    movie‘)
```

MATCH (m:‘movie‘)–[:written_by]–>(w:‘writer
‘)
WHERE toLower(d.name)=’yuriy norshteyn’
RETURN w LIMIT 200
‘‘‘

Question: the films that share actors with the film Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (m:‘movie‘)–[:starred_actors]–>(a:‘
actor‘)
MATCH (a:‘actor‘)<–[:starred_actors]–(m2:‘
movie‘)
MATCH (m2:‘movie‘)–[:release_year]–>(y:‘year
‘)
WHERE toLower(m.name)=’dil chahta hai’
AND m <> m2
RETURN y LIMIT 200
‘‘‘

D.3 Ablation experiment "All hops in one line"

Question: what films can be described by occupation
Cypher query: ‘‘‘
MATCH (movie:‘movie‘)–[:has_tags]–>(
occupationTag:‘tag‘)
WHERE toLower(occupationTag.name)=’
occupation’
RETURN movie LIMIT 200
‘‘‘

Question: which person wrote the films directed by Yuriy Norshteyn
Cypher query: ‘‘‘
MATCH (yuriyDirector:‘director‘)<–[:
directed_by]–(movie:‘movie‘)–[:written_by
]–>(writer:‘writer‘)
WHERE toLower(yuriyDirector.name)=’yuriy
norshteyn’
RETURN writer LIMIT 200
‘‘‘

Question: the films that share actors with the film Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (dilMovie:‘movie‘)–[:starred_actors
]–>(actor:‘actor‘)<–[:starred_actors]–(
otherMovie:‘movie‘)–[:release_year]–>(year
:‘year‘)
WHERE toLower(dilMovie.name)=’dil chahta
hai’
AND dilMovie <> otherMovie
RETURN year LIMIT 200
‘‘‘

D.4 Ablation experiment "One example only employing our design"

Question: the films that share actors with the film Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (dilMovie:‘movie‘)–[:starred_actors
]–>(actor:‘actor‘)
MATCH (actor:‘actor‘)<–[:starred_actors]–(
otherMovie:‘movie‘)
MATCH (otherMovie:‘movie‘)–[:release_year
]–>(year:‘year‘)
WHERE toLower(dilMovie.name)=’dil chahta
hai’
AND dilMovie <> otherMovie
RETURN year LIMIT 200
‘‘‘

D.5 Ablation experiment "Chain direction not maintained"

Question: what films can be described by occupation
Cypher query: ‘‘‘
MATCH (movie:‘movie‘)–[:has_tags]–>(
occupationTag:‘tag‘)
WHERE toLower(occupationTag.name)=’
occupation’
RETURN movie LIMIT 200
‘‘‘

Question: which person wrote the films directed by Yuriy Norshteyn
Cypher query: ‘‘‘
MATCH (movie:‘movie‘)–[:directed_by]–>(
yuriyDirector:‘director‘)
MATCH (movie:‘movie‘)–[:written_by]–>(writer
:‘writer‘)
WHERE toLower(yuriyDirector.name)=’yuriy
norshteyn’
RETURN writer LIMIT 200
‘‘‘

Question: the films that share actors with the film Dil Chahta Hai were released in which years
Cypher query: ‘‘‘
MATCH (dilMovie:‘movie‘)–[:starred_actors
]–>(actor:‘actor‘)

MATCH (otherMovie:‘movie‘)–[:starred_actors]–>(actor:‘actor‘)
MATCH (otherMovie:‘movie‘)–[:release_year]–>(year:‘year‘)
WHERE toLower(dilMovie.name)=’dil chahta hai’
AND dilMovie <> otherMovie
RETURN year LIMIT 200
‘‘‘

E Transferring methodology to SPARQL

E.1 Full prompt to transfer few-shot style

You are an expert at graph languages like CYPHER and SPARQL. You want rewrite graph queries so that each query is more readable and understandable. An example is given below:

OLD QUERY
MATCH (yr:‘year‘)<–[:release_year]–(m:‘movie ‘)–[:starred_actors]–>(a:‘actor‘)<–[: starred_actors]–(m2:‘movie‘ {{name: ’Dil Chahta Hai’}})
WHERE m <> m2
RETURN yr LIMIT 200

NEW QUERY
MATCH (dilMovie:‘movie‘)–[:starred_actors]–>(actor:‘actor‘)
MATCH (actor:‘actor‘)<–[:starred_actors]–(otherMovie:‘movie‘)
MATCH (otherMovie:‘movie‘)–[:release_year]–>(year:‘year‘)
WHERE toLower(dilMovie.name)=’dil chahta hai’
AND dilMovie <> otherMovie
RETURN year LIMIT 200

Help me rewrite the following query to make it more readable and understandable. Make sure that:

1. Each hop is articulated on a separate line to mirror the logical sequence of traversals, strictly adhering to the correct order of entities and relationships encountered
2. Maintain an unbroken logical chain where the endpoint of one hop is the starting point for the next, ensuring a coherent flow of entities throughout the query
3. Adopt variable names that are both illustrative and consistent, reflecting the entity type and any applicable constraints, like ’dilMovie’ to denote a ’movie entity constrained by the title ’Dil Chahta Hai’

Please help me rewrite the following query in the style discussed above.