SyntheT2C: Generating Synthetic Data for Fine-Tuning Large Language Models on the Text2Cypher Task

Anonymous ACL submission

Abstract

Integrating Large Language Models (LLMs) with existing Knowledge Graph (KG) databases presents a promising avenue for enhancing LLMs' efficacy and mitigating their "hallucinations". Given that most KGs reside in graph databases accessible solely through specialized query languages (e.g., Cypher), there exists a critical need to bridge the divide between LLMs and KG databases by automating the translation of natural language into Cypher queries (commonly termed the "Text2Cypher" task). Prior efforts tried to bolster LLMs' proficiency in Cypher generation through Supervised Fine-Tuning. However, these explorations are hindered by the lack of annotated datasets of Query-Cypher pairs, resulting from the labor-intensive and domain-specific nature of annotating such datasets. In this study, we propose **SyntheT2C**, a methodology for constructing a synthetic Query-Cypher pair dataset, comprising two distinct pipelines: (1) LLM-based prompting and (2) template-filling. SyntheT2C facilitates the generation of extensive Query-Cypher pairs with values sampled from an underlying Neo4j graph database. Subsequently, SyntheT2C is applied to two medical databases, culminating in the creation of a synthetic dataset, **MedT2C**. Comprehensive experiments demonstrate that the MedT2C dataset effectively enhances the performance of backbone LLMs on the Text2Cypher task. Both the SyntheT2C codebase and the MedT2C dataset will be released soon.

1 Introduction

Knowledge Graphs (KGs) constitute vital reservoirs of information within the Retrieval-Augmented Generation (RAG) paradigm (Lewis et al., 2020) of Large Language Models (LLMs). Distinguished from other information sources, KGs boast structured and meticulously curated data, rendering them conducive to seamless updates and rectifications. Such attributes position KGs as pivotal instruments for mitigating issues of knowledge cutoff and "hallucinations" within LLMs. Notably, KGs have long served as a core in numerous knowledge-intensive products and applications (Kertkeidkachorn et al., 2023; Cui et al., 2024; Xu et al., 2020). With the advent of LLMs, many researchers have focused on synergizing KGs with LLMs following the RAG framework. The inherent fidelity and adaptability of KGs make them practical assets for deployment in production environments, and also catapult KGs to the forefront of academic research.

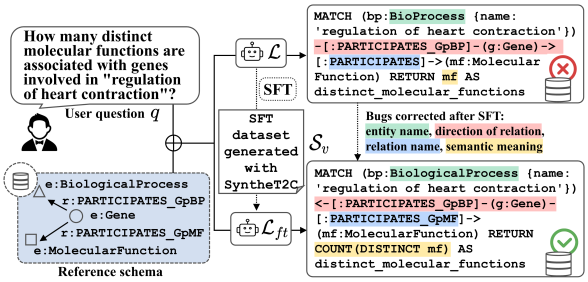

Figure 1: SyntheT2C builds synthetic data with two pipelines to SFT LLMs so that their performance on Text2Cypher task is enhanced.

While KGs represent invaluable repositories of reference information, their efficient utilization remains a formidable challenge. Early methodologies involved direct extraction of triplets from KGs, subsequently integrating these text-form triplets directly into the prompts of LLMs (Fatemi et al., 2023). However, this approach often fails to concurrently preserve both semantic and structural nuances inherent within the KG. An alternative approach involves querying existing graph databases just like human users, promising accurate and interpretable results. Nonetheless, the primary impediment lies in the LLM's ability to formulate correct and executable queries. To address this limitation, numerous query generation tools or methodologies (Zhang et al., 2022; Abdelaziz et al., 2021; Shen

et al., 2023) are proposed, aiming to translate human users' natural language queries into query languages. This task assumes paramount importance for LLM development for two pivotal reasons: (1) it empowers LLMs to consistently produce reliable queries, thereby augmenting their utilization of existing KG databases to address knowledge deficits; (2) it facilitates human interaction with KG databases through natural language, substantially lowering the barrier to entry for KG database utilization. Among the spectrum of query generation research, the sub-task of translating natural language into the Cypher (Francis et al., 2018) query language for Neo4j (Neo4j, 2012) databases stands out as a prominent research focus. This prominence is attributed to two key factors. Firstly, Neo4j is a widely adopted solution for constructing KG databases, positioning Cypher as an essential tool for accessing these extensive repositories. Secondly, Cypher is a query language specifically designed for querying graph structures, offering significantly faster performance than other query languages, such as SQL, when processing graph data. Consequently, our work centers on this subtask, commonly termed as "**T**ext**2C**ypher" (T2C).

A similar task to the Text2Cypher task is the "Text2SQL" task, wherein researchers endeavor to translate natural language sentences into SQL queries. Leveraging manually annotated datasets like SPIDER (Yu et al., 2019), numerous methodologies have emerged, including SpCQL (Guo et al., 2022) and SQLNet (Xu et al., 2017). Conversely, scant attention has been directed towards the Text2Cypher task. Existing approaches typically resort to decomposing the original query into smaller components and translating each part separately. For instance, R^3-NL2SQL (Zhou et al., 2023b) partitions the query generation process into CRUD keywords prediction, clause selection, and object type identification. Despite the success of these methods, adapting them to a specific KG database demands substantial extra effort. With the rise of LLMs, using LLMs for Cypher query generation appears promising. Notably, to the best of our knowledge, no endeavors have explored the potential application of LLMs to the Text2Cypher task. Our work aims to bridge this gap in the literature.

The Cypher writing performance of vanilla LLMs is not satisfactory. To improve it, we employ SFT, which necessitates a dataset of Question-Query pairs. However, creating such a dataset is challenging as it requires both domain-specific knowledge of the KG's content and expertise in Cypher's syntax. Consequently, there is currently no annotated dataset for the Text2Cypher task. To overcome this obstacle, we introduce SyntheT2C, a method designed to produce high-quality synthetic Question-Cypher pairs through two distinct pipelines: LLM-based prompting and template-filling (as shown in Figure 1). The LLM-based prompting pipeline aims to generate Cypher queries with greater semantic flexibility, while the template-filling pipeline focuses on producing syntactically complex Cypher queries. The generated Question-Query pairs undergo rigorous automated and manual validation, before being used to fine-tune backbone LLMs. The performance of Cypher generation is evaluated with a manually annotated evaluation dataset, complemented by a qualitative assessment using GPT as a judge. Additionally, we conduct a scalability test by fine-tuning the LLMs with larger synthetic datasets, which demonstrates that the synthetic data generated using our method does not collapse into simple patterns, thereby establishing the robustness of our approach for larger-scale applications.

SyntheT2C is tested with two medical databases: the LHY database and the Hetionet database (details in Section 4.1). The generated synthetic dataset, "MedT2C", will be made public.

In conclusion, our main contributions are:

(1) We propose the SyntheT2C framework containing two pipelines to build synthetic datasets with any Neo4j database. Our method can generate Cypher that are both grammatically correct and syntactically diverse, facilitating the construction of SFT datasets.

(2) We test and validate the effectiveness and scalability of the synthetic dataset generated with SyntheT2C. The LLMs after fine-tuning show improved Cypher writing abilities.

(3) We opensource a synthetic dataset MedT2C of optimal size, ready to be used for SFT.

2 Related works

2.1 Knowledge Graph and graph database

In recent years, KGs have emerged as fundamental resources for organizing, representing, and querying vast amounts of interconnected information or domain-specific knowledge. These graphs find applications across various domains, including but not limited to, healthcare (Cui et al., 2024; Abu-Salih et al., 2022), finance (Elhammadi et al., 2020;

Kertkeidkachorn et al., 2023), and e-commerce (Xu et al., 2020). In the realm of Natural Language Processing and Artificial Intelligence (AI), KGs serve as invaluable sources of context and factual knowledge, enabling systems to reason, infer, and generate responses with enhanced accuracy and coherence.

To handle the processing of graph-structured data, a series of graph databases were invented, including Neo4j (Neo4j, 2012), NebulaGraph (Wu et al., 2022), and Amazon Neptune (Bebee et al., 2018). Among them, our work focuses on the Neo4j database (Neo4j, 2012), a widely used graph database management system that excels in modeling and querying highly interconnected data. Neo4j database employs a powerful query language called Cypher for expressing complex graph patterns and retrieving specific data subsets.

2.2 Large Language Models

LLMs are advanced AI models that have been trained on vast amounts of text data to understand and generate human-like language. After the recent breakthrough marked by the release of InstructGPT (Ouyang et al., 2022) by OpenAI, a series of LLMs are released, featuring different advantages and drawbacks, e.g., the series of GPT models (Brown et al., 2020; OpenAI, 2023) by OpenAI, Llama (Meta, 2024) by Meta, Qwen (Bai et al., 2023) by Alibaba Cloud, InternLM (Cai et al., 2024b) by Shanghai AI Lab, etc. LLMs can comprehend and generate text across a wide range of topics and writing styles. Recent researches highlight their ability to utilize external existing tools like calculator, search engine, or databases (Patil et al., 2023; Nakano et al., 2022; Cai et al., 2024a; Qin et al., 2023). This ability is usually abstracted as "Function calling", and many of its implementations involve generating codes or queries with LLMs to interact with external tools.

2.3 Code generation

Code Generation is the process of automatically producing executable code from a higher-level representation or natural language. With the advent of LLMs, code generation has experienced a significant advancement. LLMs can now be trained on vast amounts of code and programming-related text materials, enabling them to understand and generate code snippets based on given requirements (e.g., Codex (Chen et al., 2021), Polycoder (Xu et al., 2022), and Code Llama (Rozière et al., 2024)). By leveraging the contextual understanding (Dong et al., 2023) and language capabilities of LLMs, code generation becomes more efficient, accurate, and adaptable. Code generation with LLM is not only useful in helping developers to write codes but also in providing a powerful "language" for LLM to interact with other tools: LLMs can be tuned to output executable codes or queries to manipulate external resources. This is the fundamental idea for research in "Function Calling" and Multi-Agent Systems. Current code generation methods rely on two methods for evaluation: either with automatic metrics calculated with an annotated evaluation dataset (Papineni et al., 2002; Lin, 2004; Banerjee and Lavie, 2005; Evtikhiev et al., 2023; Zhou et al., 2023a) or with comparison by a judge (human or powerful LLM like GPT-4) (Zheng et al., 2023). Both evaluation methods are used in our work.

3 Methodology

3.1 Preliminaries

The goal of the Text2Cypher task is to automatically translate a query q written in natural language to corresponding Cypher query c. With the proposed pipelines $\mathcal{P}_1$ and $\mathcal{P}_2$, a synthetic dataset $\mathcal{S}$ is built to fine-tune the backbone LLM $\mathcal{L}$. The synthetic data is generated and validated with a Neo4j database $\mathcal{B}$ and a series of automatic validators $\mathcal{V} = [\mathcal{V}_1, \mathcal{V}_2, ..., \mathcal{V}_5]$. The synthetic dataset after all the validations is denoted as $\mathcal{S}_v$. Using $\mathcal{S}_v$, $\mathcal{L}$ is fine-tuned into $\mathcal{L}_{ft}$. The Cypher queries generated by $\mathcal{L}$ (resp. $\mathcal{L}_{ft}$) are noted as c_1 (resp. c_2).

3.2 Synthetic dataset generation

Generating the synthetic dataset is not trivial because synthetic data usually has difficulty in balancing grammatical correctness, semantic correctness, node coverage, edge coverage, and Cypher complexity. As a result, we propose a method of generation with two pipelines, as illustrated in Figure 2). The LLM-based prompting pipeline ($\mathcal{P}_1$), emphasizes semantic variety, while the template-filling pipeline ($\mathcal{P}_2$), focuses on syntactic complexity. By employing these complementary pipelines, we aim to produce a synthetic dataset that captures the nuanced balance of linguistic, semantic, and structural properties.

3.2.1 LLM-based prompting pipeline

This pipeline adopts an idea similar to Knowledge Distillation: to SFT a weaker LLM, we could use

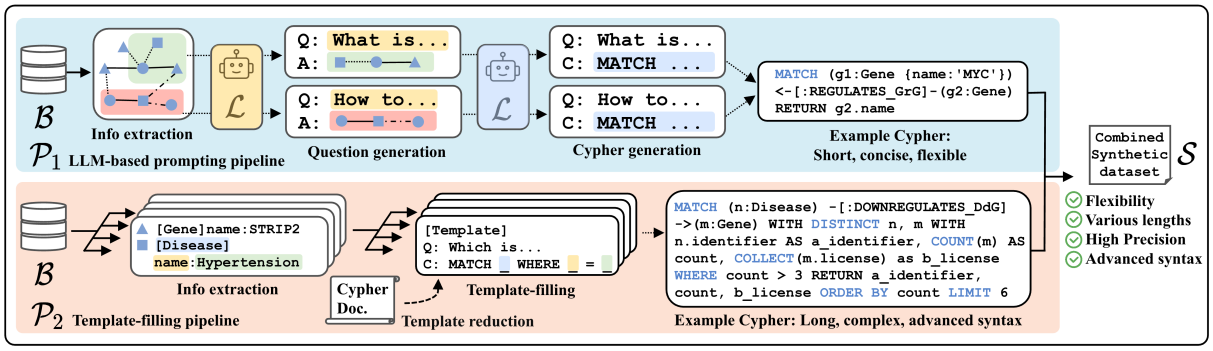

Figure 2: Workflow of two pipelines inside SyntheT2C.

the Cyphers generated by stronger LLMs. While off-the-shelf LLMs are typically not optimized for Cypher query generation, as Cypher likely constitutes only a small fraction of their pre-training data, they have nonetheless demonstrated strong in-context learning capabilities (Dong et al., 2023). Therefore, half of $\mathcal{S}$ is built by few-shot prompting GPT-4o (OpenAI, 2023). To further simplify the task of generation and to ensure a higher quality of the generated data, we split the whole generation task into (1) extracting information from the database; (2) determining the question categories; and (3) generating the Cyphers for each category with extracted information.

The workflow for the LLM-based prompting method is delineated in Figure 2 (upper part, $\mathcal{P}_1$). Initially, we commence by extracting metadata from the KG stored in the Neo4j database $\mathcal{B}$. This extraction includes sampling example nodes and edges to construct few-shot prompts, along with capturing the schema of the database to facilitate the generation of grounded Cyphers. An illustrative instance of extracted metadata is provided in Appendix A. Subsequently, this metadata serves as a foundational component in all ensuing prompts, ensuring the generation of executable Cyphers. Before initiating the Cypher generation process, a preliminary step involves prompting the LLM to propose potential question categories, thereby mitigating the risk of redundant outputs. The backbone LLM undergoes multiple iterations to propose these question categories, as detailed in the prompt showcased in Appendix B.1. These proposed categories are then consolidated to eliminate duplicates, as instructed in the prompt outlined in Appendix B.3. After the deduplication, GPT-4o is prompted to generate synthetic Question-Cypher pairs with the prompt outlined in Appendix B.2. In our experiment, we fix a list of 12 categories (referred to as `categories`) to facilitate the comparison.

3.2.2 Template-filling pipeline

The second pipeline of Cypher generation adopts the template-filling method, a classic approach in code generation known for its flexible output and potentially complex syntax. We introduce this pipeline as a complement to the first one, leveraging manually crafted templates to generate Cyphers with more advanced syntax, thereby enabling backbone $\mathcal{L}$ to solve complicated questions.

In this pipeline, depicted in Figure 2 (lower part, $\mathcal{P}_2$), numerous templates are initially manually authored. Subsequently, actual values from different fields are sampled from the Neo4j database $\mathcal{B}$ to populate these templates, resulting in the generation of complete executable Cypher queries.

One such template is illustrated in Figure 4. In this example, the `subschema` is introduced to manage cases where the entire database cannot be loaded at once, necessitating the selection and injection of only the relevant subgraph into the prompt. The variables `label_i` and `prop_j` represent the randomly sampled names of nodes and their attributes. These templates are initially crafted taking inspiration from *Cypher Generator* (Onofrei, 2024), then enriched and verified by the authors.

Once these templates are established, synthetic Cyphers with complex syntax can be effortlessly generated. However, it is important to note that crafting and validating these templates require considerable time and effort.

3.3 Quality validation

To ensure the quality of the generated synthetic Question-Cypher pairs before their application in SFT, it's imperative to conduct thorough validation to prevent the "garbage in, garbage out" scenario. However, manually scrutinizing thousands of Cypher queries is arduous and time-consuming. In response, a suite of automatic validators has been implemented to alleviate the burden of manual in-

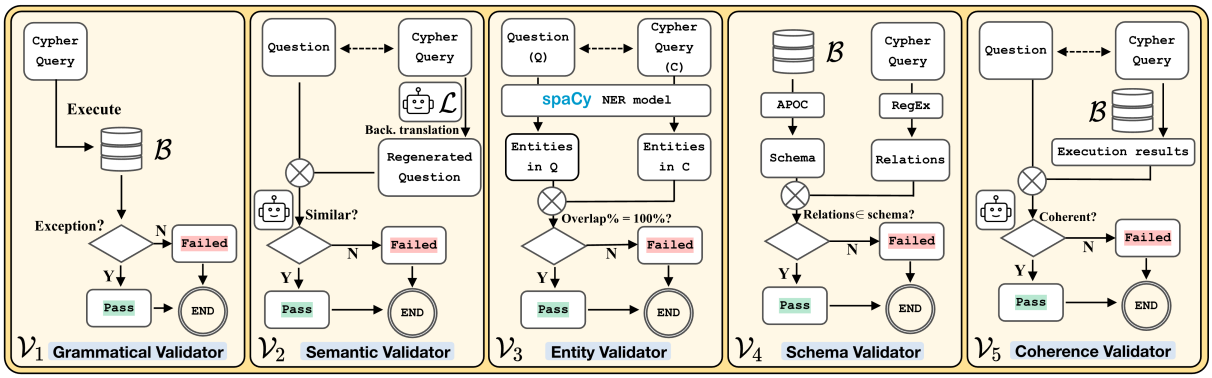

Figure 3: Illustration of the automatic validators.

```
def prompter(label_1, prop_1, prop_2):
    subschema = get_subgraph_schema(jschema,
          [label_1], 2, True)
    message = {
        "prompt": "Convert the following question
              into a Cypher query using the provided
              graph schema!",
        "question": f"""Find all {prop_1} for
              {label_1} that have {prop_2} after
              January 1, 2020!""",
        "schema": f"Graph schema: {subschema}",
        "cypher": f"MATCH (n:{label_1}) WHERE
              date(n.{prop_2}) > date('2020-01-01')
              RETURN n.{prop_1}"
        }
    return message
```

Figure 4: Example template in Template-filling pipeline.

spection. In the end, the Cyphers that pass through these automated validators undergo a final round of meticulous manual validation by researchers.

3.3.1 Automatic validation

We propose five automatic validators: the Grammatical Validator, Semantic Validator, Entity Validator, Schema Validator, and Coherence Validator, each playing a crucial role in ensuring the integrity of the generated synthetic data. These validators' fundamental concepts are illustrated in Figure 3. The LLM used in the validators is GPT-3.5-Turbo.

The **Grammatical Validator** validates the syntax correctness of each Cypher in $\mathcal{S}$ by executing them in the deployed graph database $\mathcal{B}$. If a Cypher is executed without encountering any "Error/Exceptions", it is deemed to have passed this validation.

The design of **Semantic Validator** is inspired by the research in machine translation (Hoang et al., 2018). This validator utilizes an LLM to translate the generated Cypher back into a natural language question. It then computes the semantic similarity between the translated question and the original question. If the similarity score exceeds a predefined threshold, the Cypher passes validation. We also implement an alternative version of the Semantic Validator, where the LLM assesses semantic similarity directly. Both versions produce coherent validation results, with the latter being adopted for efficiency in subsequent experiments. The prompt used in this validator is presented in Appendix C.1.

The **Entity Validator** assesses the coverage of entities in the generated Cyphers. The entities in the original question q are extracted via Named Entity Recognition (NER) using the spaCy (Honnibal and Montani, 2017) model `en_core_web_sm`. Entities in the generated Cypher c are parsed and extracted using Regular Expressions. A successful validation requires 100% coverage of q's entities in c. English entities are first transformed into lemmas using spaCy for fuzzy matching.

Subsequently, the **Schema Validator** ensures the correctness of relations in the generated Cyphers. Relations in c are extracted via Regular Expressions and validated against the schema of $\mathcal{B}$. A Cypher passes this validation only when all contained relations are valid edges.

Lastly, the **Coherence Validator** executes the Cypher against $\mathcal{B}$ and evaluates the coherence between the execution results and the original question with LLM, using the prompt presented in Appendix C.2.

In the end, only Cyphers that have passed all validations proceed to manual validation.

3.3.2 Manual validation

Each Cypher checked by the validators is randomly assigned to two researchers, who independently assess its quality. If both researchers provide a unanimous judgment, their consensus is adopted. In cases of divergent opinions, a third researcher is brought in for further review. The final validation outcome for such Cyphers is determined through a majority vote among the three researchers.

4 Experiments

4.1 LHY and Hetionet Graph databases

Throughout our experiment, we employed two Neo4j databases housing general medical knowledge in graph form: the LHY Medical Knowledge Database (referred to as "LHY") and the Hetionet Medical Knowledge Database (referred to as "Hetionet"). Both databases are publicly accessible, differing primarily in language: the data within the LHY database is presented in Chinese, whereas Hetionet is written in English.

The LHY Database (Liu, 2018) serves as the backend database for a Medical Question-Answering system. This database comprises comprehensive medical knowledge, encompassing a wide array of diseases, symptoms, drugs, and related information. Its content is sourced from medical websites, meticulously cleaned, reorganized, and stored within a Neo4j database. There are about 44k entities and 300k relations in it.

Hetionet (Himmelstein et al., 2017) is an open and free-to-use database of biomedical knowledge resource implementing "hetnet" model. Aggregating insights from 29 public databases, Hetionet boasts a knowledge network spanning various fields, encompassing a wide array of entities, including genes, compounds, anatomical structures, diseases, symptoms, side effects, etc. There are approximately 47k entities and 2.2 million relations in the Hetionet database.

The detailed statistics of both databases are presented in Appendix D.

4.2 Evaluation dataset and metrics

We utilize a dataset comprising 300 manually annotated and verified samples to evaluate our experiments. This dataset includes 150 questions annotated based on the Hetionet and LHY databases, respectively. Take Hetionet as an example, for every category among the 12 categories generated in Section 3.2.1, we employ GPT-3.5-Turbo to generate 10 new questions, forming 120 "in-domain" questions. Additionally, we introduce 3 unseen categories and generate 10 new questions for each new category, totaling 30 "out-of-domain" questions. For each of the 300 questions, the authors write a ground-truth Cypher query, which is then executed against the two databases to get the ground-truth execution results.

This annotated dataset allows us to evaluate two aspects of LLMs' Cypher generating performance:

(1) **Cypher quality**, which is crucial if the generated Cypher is integrated into larger systems;

(2) **Execution result accuracy**, to gauge the quality of the output for end users.

4.2.1 Evaluation of Cypher quality

The backbone LLMs, both pre-SFT and post-SFT, are tasked with generating Cyphers for the 300 questions in the evaluation dataset. Using GPT-4o (OpenAI, 2023), we determine the superior Cypher from the two provided versions. For each pair of Cyphers, we conduct two evaluations by varying the order of presenting the Cypher queries in the prompt to mitigate order-induced bias. If evaluations of both orders yield identical results, this judgment is accepted as the final outcome; otherwise, it is deemed a "Draw".

4.2.2 Evaluation of execution result accuracy

The generated Cyphers c_2 are executed on database $\mathcal{B}$ to get execution results res_{gen}. Then the accuracy (acc) is calculated with the ground-truth execution results res_{gt} like this:

$$acc = \frac{\#(res_{gen} \cap res_{gt})}{\#(res_{gen})}, \quad (1)$$

where $\#(.)$ calculates the cardinality of a set.

4.3 Experiment setup

4.3.1 Cypher LLMs

Extensive experiments are conducted with four LLMs, including open/closed-source models. For open-source models, we evaluate Llama3, Qwen2 and InternLM2. For closed-sources model, we test GPT. The exact versions of the backbone LLMs are listed in Appendix E.

4.3.2 Supervised Fine-Tuning

We utilize Low-Rank Adaptation (LoRA) to fine-tune the vanilla LLMs. Specifically, the open-source models are trained for 6 epochs with a linear scheduler, starting at a learning rate of 1e-6. AdamW is used as the optimizer, and the training batch size is 6. The fine-tuning of GPT is facilitated by its official API. The experiments on all LLMs, are conducted on Nvidia GeForce 4090 GPU. All the experiments totaled about 1100 GPU hours.

4.4 Supervised Fine-Tuning experiments

The backbone LLMs are fine-tuned with the MedT2C dataset, comprising 750 samples generated with the two pipelines and two Neo4j

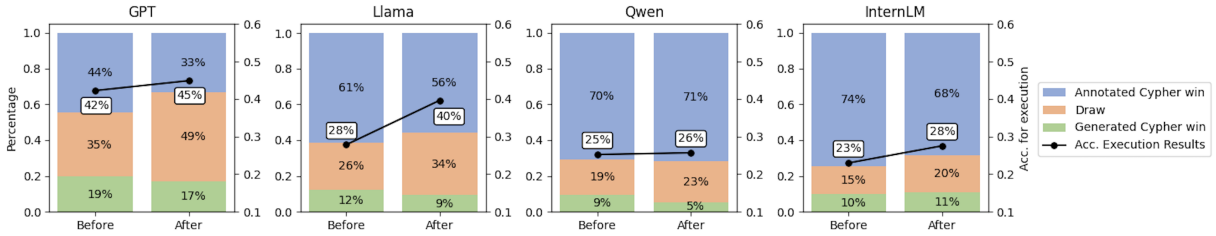

Figure 5: Result of Supervised Fine-Tuning each LLMs with MedT2C. Accuracy annotations marked in white box.

databases, totaling 3000 samples. The MedT2C dataset contains high-quality Question-Cypher pairs that passed all the automatic validations as well as the manual validation. In Appendix F we report the passing rate of each validator as a guide for further improvement of MedT2C's data quality.

A list of LLMs including GPT, Llama, Qwen, and InternLM are fine-tuned using MedT2C. We evaluated the change in Cypher writing performance of these LLMs, and the results are shown in Figure 5. The results show that MedT2C helps the LLM to produce more Cypher queries that are on par with or better than the human annotated ones.

In Figure 5, the win rates are calculated in comparison with the ground-truth Cyphers. We further conduct an experiment to compare directly the c_1 and c_2 generated with the same LLM with GPT-4o, using the prompt presented in Appendix G. The comparison results are shown in Figure 6. From these results, we can conclude that while the improvement may appear minor when comparing against the ground-truth Cyphers, a visible enhancement in Cypher quality is evident when comparing to the Cyphers generated by the pre-SFT model. We explain this difference as follows: the human annotations have a far higher quality than the Cyphers generated by vanilla LLMs. Therefore, even though the LLMs are enhanced after SFT, their output is still inferior to the human-annotated Cyphers, which is why the evaluation results in Figure 5 seem largely unchanged.

4.5 Scaling experiments

In this section, we test the scalability of our pipeline for generating synthetic data. We rerun the data generation pipelines to create scaled versions that are 1/16, 1/4, 4, 8, and 16 times the size of the original MedT2C. Vanilla LLMs are then fine-tuned with these scaled datasets. The results are reported in Figure 7. These results demonstrate that, up to the size of the MedT2C dataset, increasing the size of the synthetic dataset leads to better performance, especially in terms of Cypher Quality.

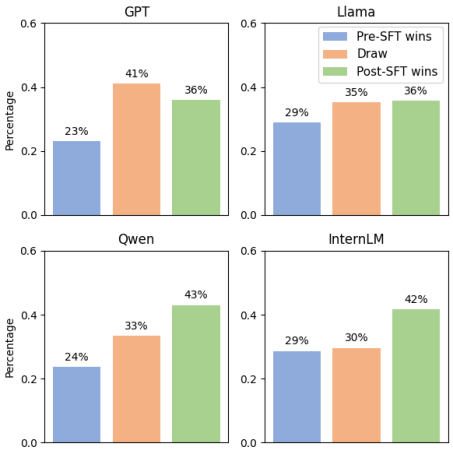

Figure 6: Impact of SFT on each LLM. The Cypher generated with pre-SFT and post-SFT LLMs are compared directly with GPT-4o.

However, once the size exceeds that of MedT2C, further increasing the dataset size results in either marginal improvements or decreases. Based on this experiment, we determine the optimal size for the published MedT2C dataset (highlighted in red), as it balances efficiency and performance.

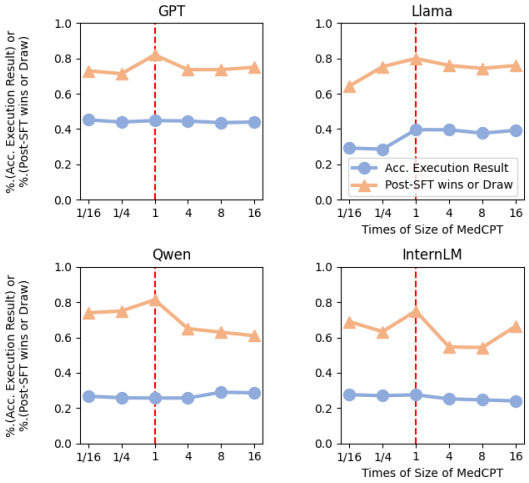

Figure 7: Plots of scaling test's results.

4.6 Ablation experiments

To evaluate the efficacy of each component introduced, we conduct a series of ablation experiments.

First, we test the pipelines by running SFT experiments using only the data generated by each pipeline individually. We then verify the effectiveness of each automatic validator by evaluating them in isolation, using only one validator at a time. Since each component is designed to be modular and independent, we adopt this mode of ablation, rather than removing the components one by one from the complete setting, to emphasize the increment brought by each component separately. For both ablation tests, the backbone LLM is fixed as Llama3. The dataset size is set to be the same as MedT2C (3000 in total). The experiments results are reported in Table 1 and Table 2 respectively. Here the Cypher Quality is calculated with respect to ground-truth Cyphers.

Settings	Cypher Quality	Result Acc.
Pre-SFT	38.67%(–)	27.83%(–)
LHY-LLM	41.67%(+3.00)	27.86%(+0.03)
LHY-Temp.	34.67%(-4.00)	26.54%(-1.29)
Hetionet-LLM	42.83%(+4.16)	33.09%(+5.26)
Hetionet-Temp.	36.00%(-2.67)	26.68%(-1.15)
All (MedT2C)	**44.00%**(+5.33)	**39.65%**(+11.82)

Table 1: Results of pipeline ablation test.

As presented in Table 1, when we use only the data generated by the template-filling pipeline to SFT the Llama3 model, the model's performance actually declined after SFT. This can be attributed to the design of the template-filling pipeline, which emphasizes generating syntactically complex Cypher queries. When SFT is performed using only this data, the backbone LLM tends to produce unnecessarily complicated Cypher queries (e.g., breaking one query into two and then merging them). While this ability to write more complex Cypher queries is not directly reflected in the evaluation metrics, as the "hard questions" requiring advanced syntactic knowledge constitute only a small portion of the evaluation dataset, such data can enhance the LLM's generalization capacity when combined with data from the LLM-based prompting pipeline.

Settings	Cypher Quality	Result Acc.
Pre-SFT	38.67%(–)	27.83%(–)
No validator	38.34%(-0.33)	27.96%(+0.13)
✓Grammar V.	38.34%(-0.33)	28.95%(+1.12)
✓Semantic V.	43.67%(+5.00)	31.65%(+3.82)
✓Entity V.	40.00%(+1.33)	28.03%(+0.20)
✓Schema V.	42.00%(+3.33)	26.11%(-1.72)
✓Coherence V.	41.33%(+2.66)	32.05%(+4.22)
All (MedT2C)	**44.00%**(+5.33)	**39.65%**(+11.82)

Table 2: Results of validator ablation test.

As shown in Table 2, each individual validator contributes some improvement, either in terms of Cypher quality or the accuracy of the execution results. Notably, the combination of all five validators yields the most significant increase in performance. This can be attributed to the validators' collective ability to mitigate the majority of the bugs in the SFT dataset, thereby enhancing the overall quality of the generated Cypher queries.

5 Limitations

The primary limitation of our work is the challenge in writing the templates. While the templates are designed to be independent from the base Neo4j databases, some adaptation work is still necessary when applying them to new Neo4j databases. Additionally, writing new templates is time-consuming, making the expansion of the current template library difficult. Furthermore, a significant number of the generated Cypher queries are directly filtered out during the construction of MedT2C, resulting in a waste of resources. Developing methods to quickly fix Cyphers that do not satisfy all the validation criteria, instead of simply regenerating more Cyphers, could help reduce the carbon footprint of SyntheT2C.

6 Potential risks

Even though SyntheT2C is designed to automatically generate synthetic datasets, its usage requires close monitoring and manual validation to prevent the inadvertent inclusion of private or sensitive information. Additionally, the post-SFT LLM should be used with caution. Despite improvements in their Cypher generation performance, there remains a slight risk of producing embedded Cyphers that could lead to issues such as Out-of-Memory.

7 Conclusion

We present SyntheT2C, a comprehensive framework to generate synthetic data for SFT various LLMs on the Text2Cypher task. Our approach encompasses dataset construction, data validation, and SFT evaluation, providing a reference framework for future research in the Cypher-related field. Additionally, our findings confirm the effectiveness of synthetic data, suggesting that similar techniques can address problems where annotation is difficult or insufficient. Finally, we will also open-source the MedT2C dataset, aiming to contribute to the technical advancements in relevant topics.

References

Ibrahim Abdelaziz, Srinivas Ravishankar, Pavan Kapanipathi, Salim Roukos, and Alexander Gray. 2021. A semantic parsing and reasoning-based approach to knowledge base question answering. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(18):15985–15987.

Bilal Abu-Salih, Muhammad AL-Qurishi, Mohammed Alweshah, Mohammad AL-Smadi, Reem Alfayez, and Heba Saadeh. 2022. Healthcare knowledge graph construction: State-of-the-art, open issues, and opportunities. *Preprint*, arXiv:2207.03771.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, Jianxin Ma, Rui Men, Xingzhang Ren, Xuancheng Ren, Chuanqi Tan, Sinan Tan, Jianhong Tu, Peng Wang, Shijie Wang, Wei Wang, Shengguang Wu, Benfeng Xu, Jin Xu, An Yang, Hao Yang, Jian Yang, Shusheng Yang, Yang Yao, Bowen Yu, Hongyi Yuan, Zheng Yuan, Jianwei Zhang, Xingxuan Zhang, Yichang Zhang, Zhenru Zhang, Chang Zhou, Jingren Zhou, Xiaohuan Zhou, and Tianhang Zhu. 2023. Qwen technical report. *Preprint*, arXiv:2309.16609.

Satanjeev Banerjee and Alon Lavie. 2005. METEOR: An automatic metric for MT evaluation with improved correlation with human judgments. In *Proceedings of the ACL Workshop on Intrinsic and Extrinsic Evaluation Measures for Machine Translation and/or Summarization*, pages 65–72, Ann Arbor, Michigan. Association for Computational Linguistics.

Bradley R. Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Michael Personick, K. Jeric Rajan, Simone Rondelli, Alexander Ryazanov, Michael Schmidt, Kunal Sengupta, Bryan B. Thompson, Divij Vaidya, and Shawn Xiong Wang. 2018. Amazon neptune: Graph data management in the cloud. In *International Workshop on the Semantic Web*, pages 1–2.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language models are few-shot learners. *Preprint*, arXiv:2005.14165.

Tianle Cai, Xuezhi Wang, Tengyu Ma, Xinyun Chen, and Denny Zhou. 2024a. Large language models as tool makers. *Preprint*, arXiv:2305.17126.

Zheng Cai, Maosong Cao, Haojiong Chen, Kai Chen, Keyu Chen, Xin Chen, Xun Chen, Zehui Chen, Zhi Chen, Pei Chu, Xiaoyi Dong, Haodong Duan, Qi Fan, Zhaoye Fei, Yang Gao, Jiaye Ge, Chenya Gu, Yuzhe Gu, Tao Gui, Aijia Guo, Qipeng Guo, Conghui He, Yingfan Hu, Ting Huang, Tao Jiang, Penglong Jiao, Zhenjiang Jin, Zhikai Lei, Jiaxing Li, Jingwen Li, Linyang Li, Shuaibin Li, Wei Li, Yining Li, Hongwei Liu, Jiangning Liu, Jiawei Hong, Kaiwen Liu, Kuikun Liu, Xiaoran Liu, Chengqi Lv, Haijun Lv, Kai Lv, Li Ma, Runyuan Ma, Zerun Ma, Wenchang Ning, Linke Ouyang, Jiantao Qiu, Yuan Qu, Fukai Shang, Yunfan Shao, Demin Song, Zifan Song, Zhihao Sui, Peng Sun, Yu Sun, Huanze Tang, Bin Wang, Guoteng Wang, Jiaqi Wang, Jiayu Wang, Rui Wang, Yudong Wang, Ziyi Wang, Xingjian Wei, Qizhen Weng, Fan Wu, Yingtong Xiong, Chao Xu, Ruiliang Xu, Hang Yan, Yirong Yan, Xiaogui Yang, Haochen Ye, Huaiyuan Ying, Jia Yu, Jing Yu, Yuhang Zang, Chuyu Zhang, Li Zhang, Pan Zhang, Peng Zhang, Ruijie Zhang, Shuo Zhang, Songyang Zhang, Wenjian Zhang, Wenwei Zhang, Xingcheng Zhang, Xinyue Zhang, Hui Zhao, Qian Zhao, Xiaomeng Zhao, Fengzhe Zhou, Zaida Zhou, Jingming Zhuo, Yicheng Zou, Xipeng Qiu, Yu Qiao, and Dahua Lin. 2024b. Internlm2 technical report. *Preprint*, arXiv:2403.17297.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde de Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, Alex Ray, Raul Puri, Gretchen Krueger, Michael Petrov, Heidy Khlaaf, Girish Sastry, Pamela Mishkin, Brooke Chan, Scott Gray, Nick Ryder, Mikhail Pavlov, Alethea Power, Lukasz Kaiser, Mohammad Bavarian, Clemens Winter, Philippe Tillet, Felipe Petroski Such, Dave Cummings, Matthias Plappert, Fotios Chantzis, Elizabeth Barnes, Ariel Herbert-Voss, William Hebgen Guss, Alex Nichol, Alex Paino, Nikolas Tezak, Jie Tang, Igor Babuschkin, Suchir Balaji, Shantanu Jain, William Saunders, Christopher Hesse, Andrew N. Carr, Jan Leike, Josh Achiam, Vedant Misra, Evan Morikawa, Alec Radford, Matthew Knight, Miles Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021. Evaluating large language models trained on code. *Preprint*, arXiv:2107.03374.

Hejie Cui, Jiaying Lu, Shiyu Wang, Ran Xu, Wenjing Ma, Shaojun Yu, Yue Yu, Xuan Kan, Chen Ling, Tianfan Fu, Liang Zhao, Joyce Ho, Fei Wang, and Carl Yang. 2024. A review on knowledge graphs for healthcare: Resources, applications, and promises. *Preprint*, arXiv:2306.04802.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Zhiyong Wu, Baobao Chang, Xu Sun, Jingjing Xu, Lei Li, and Zhifang Sui. 2023. A survey on in-context learning. *Preprint*, arXiv:2301.00234.

Sarah Elhammadi, Laks V.S. Lakshmanan, Raymond Ng, Michael Simpson, Baoxing Huai, Zhefeng Wang, and Lanjun Wang. 2020. A high precision

pipeline for financial knowledge graph construction. In *Proceedings of the 28th International Conference on Computational Linguistics*, pages 967–977, Barcelona, Spain (Online). International Committee on Computational Linguistics.

Mikhail Evtikhiev, Egor Bogomolov, Yaroslav Sokolov, and Timofey Bryksin. 2023. Out of the bleu: How should we assess quality of the code generation models? *Journal of Systems and Software*, 203:111741.

Bahare Fatemi, Jonathan Halcrow, and Bryan Perozzi. 2023. Talk like a graph: Encoding graphs for large language models. *Preprint*, arXiv:2310.04560.

Nadime Francis, Alastair Green, Paolo Guagliardo, Leonid Libkin, Tobias Lindaaker, Victor Marsault, Stefan Plantikow, Mats Rydberg, Petra Selmer, and Andrés Taylor. 2018. Cypher: An evolving query language for property graphs. In *Proceedings of the 2018 International Conference on Management of Data*, SIGMOD '18, page 1433–1445, New York, NY, USA. Association for Computing Machinery.

Aibo Guo, Xinyi Li, Guanchen Xiao, Zhen Tan, and Xiang Zhao. 2022. Spcql: A semantic parsing dataset for converting natural language into cypher. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, CIKM '22, page 3973–3977, New York, NY, USA. Association for Computing Machinery.

Daniel Scott Himmelstein, Antoine Lizee, Christine Hessler, Leo Brueggeman, Sabrina L Chen, Dexter Hadley, Ari Green, Pouya Khankhanian, and Sergio E Baranzini. 2017. Systematic integration of biomedical knowledge prioritizes drugs for repurposing. *eLife*, 6:e26726.

Vu Cong Duy Hoang, Philipp Koehn, Gholamreza Haffari, and Trevor Cohn. 2018. Iterative back-translation for neural machine translation. In *Proceedings of the 2nd Workshop on Neural Machine Translation and Generation*, pages 18–24, Melbourne, Australia. Association for Computational Linguistics.

Matthew Honnibal and Ines Montani. 2017. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear.

Natthawut Kertkeidkachorn, Rungsiman Nararatwong, Ziwei Xu, and Ryutaro Ichise. 2023. Finkg: A core financial knowledge graph for financial analysis. In *2023 IEEE 17th International Conference on Semantic Computing (ICSC)*, pages 90–93.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2020. Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Advances in Neural Information Processing Systems*, volume 33, pages 9459–9474. Curran Associates, Inc.

Chin-Yew Lin. 2004. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, Barcelona, Spain. Association for Computational Linguistics.

Huanyong Liu. 2018. Question answering system on medical knowledge graph.

Meta. 2024. Introducing meta llama 3: The most capable openly available llm to date. *Meta document*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, Xu Jiang, Karl Cobbe, Tyna Eloundou, Gretchen Krueger, Kevin Button, Matthew Knight, Benjamin Chess, and John Schulman. 2022. Webgpt: Browser-assisted question-answering with human feedback. *Preprint*, arXiv:2112.09332.

Neo4j. 2012. Neo4j - the world's leading graph database.

Silvia Onofrei. 2024. Cypher generation: The good, the bad and the messy.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744. Curran Associates, Inc.

Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. 2002. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics*, pages 311–318. Association for Computational Linguistics.

Shishir G. Patil, Tianjun Zhang, Xin Wang, and Joseph E. Gonzalez. 2023. Gorilla: Large language model connected with massive apis. *Preprint*, arXiv:2305.15334.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. 2023. Toolllm: Facilitating large language models to master 16000+ real-world apis. *Preprint*, arXiv:2307.16789.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez,

Jade Copet, Faisal Azhar, Hugo Touvron, Louis Martin, Nicolas Usunier, Thomas Scialom, and Gabriel Synnaeve. 2024. Code llama: Open foundation models for code. *Preprint*, arXiv:2308.12950.

Ran Shen, Gang Sun, Hao Shen, Yiling Li, Liangfeng Jin, and Han Jiang. 2023. Spsql: Step-by-step parsing based framework for text-to-sql generation. *Preprint*, arXiv:2305.11061.

Min Wu, Xinglu Yi, Hui Yu, Yu Liu, and Yujue Wang. 2022. Nebula graph: An open source distributed graph database. *Preprint*, arXiv:2206.07278.

Da Xu, Chuanwei Ruan, Evren Korpeoglu, Sushant Kumar, and Kannan Achan. 2020. Product knowledge graph embedding for e-commerce. In *Proceedings of the 13th International Conference on Web Search and Data Mining*, WSDM '20. ACM.

Frank F. Xu, Uri Alon, Graham Neubig, and Vincent J. Hellendoorn. 2022. A systematic evaluation of large language models of code. *Preprint*, arXiv:2202.13169.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *Preprint*, arXiv:1711.04436.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *Preprint*, arXiv:1809.08887.

Minhao Zhang, Ruoyu Zhang, Yanzeng Li, and Lei Zou. 2022. Crake: Causal-enhanced table-filler for question answering over large scale knowledge base. *Preprint*, arXiv:2207.03680.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric. P Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Shuyan Zhou, Uri Alon, Sumit Agarwal, and Graham Neubig. 2023a. CodeBERTScore: Evaluating code generation with pretrained models of code. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 13921–13937, Singapore. Association for Computational Linguistics.

Yuhang Zhou, He Yu, Siyu Tian, Dan Chen, Liuzhi Zhou, Xinlin Yu, Chuanjun Ji, Sen Liu, Guangnan Ye, and Hongfeng Chai. 2023b. r^3-nl2gql: A hybrid models approach for for accuracy enhancing and hallucinations mitigation. *Preprint*, arXiv:2311.01862.

A Example of extracted KG information

Here we present the information (metadata) extracted from the KG database "Hetionet" in Figure 8. We stored the metadata of the KG, including the node properties, the relationship properties, and the valid relationships. This information is integrated in the following prompts to ensure that the LLM output is correct Cyphers. In other prompts, this metadata is referred to as `schema`.

B Prompts for LLM-based prompting pipeline

In this appendix, we present all the prompts we used in the LLM-based prompting pipeline.

B.1 Prompt to propose categories of questions

In Figure 9 we show the prompt used to propose candidate categories of questions. We decided to first generate categories of questions instead of generating directly the questions because this practice helps reduce duplicated questions.

B.2 Prompt to generate questions for each category

This prompt presented in Figure 11 is used to generate questions in natural language for each proposed `category`. This prompt includes few-shot examples to help ensure the output Cypher follows the format requirements.

B.3 Prompt to merge categories of questions

The prompt presented in Figure 10 is used to merge the previously generated categories. The merged and de-duplicated list of categories is then stored and will be referred to as `category` in later prompts.

C Prompts used in automatic validators

C.1 Prompt of Semantic Validator

Here we present the prompt used in the Semantic Validator in Figure 12. The `schema` mentioned in this prompt is the metadata presented in Appendix 8. The `example` represents the few-shot examples written by the authors, here we show the English example for the Hetionet database in Figure 13. Lastly, the `json_object` in the prompt contains the question and the Cypher query to be evaluated.

C.2 Prompt of Coherence Validator

In this appendix, we present the prompt used in the Coherence Validator in Figure 14. Similar to other prompts, we provided few-shot examples in this prompt. The `question` and `results` in the prompt are the original question and execution results used as the input for this validation.

D Important statistics of the LHY and the Hetionet databases

Here we present the important statistics of the LHY database in Table 3 and Table 4, including the examples of nodes and entities inside this database. The examples in both tables are translated from Chinese to English. Similarly, the important statistics of the Hetionet database with examples of nodes and entities are grouped in Table 5 and Table 4.

E Exact versions of the backbone LLMs

The exact versions of the LLMs used in our experiments are listed in Table 7. Except GPT-3.5-Turbo, the backbone LLMs are deployed locally using the versions available on HuggingFace.

F Passing rate of MedT2C for each automatic validator

The passing rate of MedT2C dataset for each automatic validator is reported in Table 8. The LLM used in the Semantic Validator and the Coherence Validator is GPT-3.5-Turbo. These two validators are not run on the LLM-based prompting pipeline because this pipeline uses GPT-4o. Given that GPT-4o is more powerful than GPT-3.5-Turbo, it is not accurate to evaluate its output with GPT-3.5-Turbo, nor with GPT-4o itself. Besides, noted that the passing rate of Coherence Validator is especially low compared to other passing rate. This is because for Coherence Validator specifically, the samples that failed any one of the previous validators is judged as `False` directly to save the calling of GPT API. Therefore the passing rate of Coherence Validator reported here is lower than the actual one, but it does not affect the "All passed" ratio.

G Prompts used for Cypher quality evaluation

We use GPT-4o to judge the quality of two versions of Cypher queries corresponding to the same set of questions written in natural language. The prompt used for this part is shown in Figure 15. We provide different aspects of evaluation and ask GPT-4o to give detailed reasons when evaluating, because these techniques bring more accurate evaluation results in practice.

```
Node properties are the following:
Disease {easy_get: STRING, cure_lasttime: STRING, cured_prob: STRING, name: STRING, desc:
    STRING, prevent: STRING, cure_way: LIST, cause: STRING, cure_department: LIST},Drug
    {name: STRING},Food {name: STRING},Check {name: STRING},Department {name:
    STRING},Producer {name: STRING},Symptom {name: STRING}

Relationship properties are the following:
recommand_eat {name: STRING}, no_eat {name: STRING}, do_eat {name: STRING}, belongs_to {name:
    STRING}, common_drug {name: STRING}, drugs_of {name: STRING}, recommand_drug {name:
    STRING}, need_check {name: STRING}, has_symptom {name: STRING}, acompany_with {name:
    STRING}

The relationships are the following:
(:Disease)-[:belongs_to]->(:Department), (:Disease)-[:common_drug]->(:Drug),
    (:Disease)-[:recommand_drug]->(:Drug), (:Disease)-[:need_check]->(:Check),
    (:Disease)-[:has_symptom]->(:Symptom), (:Disease)-[:acompany_with]->(:Disease),
    (:Disease)-[:recommand_eat]->(:Food), (:Disease)-[:no_eat]->(:Food),
    (:Disease)-[:do_eat]->(:Food), (:Department)-[:belongs_to]->(:Department),
    (:Producer)-[:drugs_of]->(:Drug)
```

Figure 8: The metadata extracted from the Hetionet database.

```
You are an experienced and useful Python and Neo4j/Cypher developer.

I have a knowledge graph for which I would like to generate interesting questions that span
    12 categories (or types) about the graph. They should cover single-node questions, two
    or three more nodes, relationships, and paths. Please suggest 12 categories together
    with their short descriptions. Here is the graph schema:

{schema}
```

Figure 9: The prompt used to generate categories of questions.

```
You are an experienced doctor and you have a knowledge graph for which you would like to
    generate interesting questions of 12 categories.

Here are some candidate categories:

{categories_list}.

You should merge similar categories and remove the duplicates. Finally, give me a short
    description of each category.
```

Figure 10: The prompt used to merge proposed categories.

```
You are an experienced Cypher developer and English-speaking doctor and a helpful assistant
    designed to output JSON

Generate {k} questions and their corresponding Cypher statements about the Neo4j graph
    database with the following schema:

{schema}

The questions should cover {category} and should be phrased in a natural conversational
    manner. Make the questions diverse and interesting. Make sure to use the latest Cypher
    version and that all the queries are working Cypher queries for the provided graph. You
    may add values for the node attributes as needed. Do not add any comments, do not label
    or number the questions.

Here are some examples of the Question-Cypher pairs to be generated:

"question": "What are the diseases that commonly accompany 'Depression'?",
"cypher": "MATCH (d1:Disease {{name:'Depression'}}) -[:acompany_with]-> (d2:Disease) RETURN
    d2.name"

"question": "Can you list diseases that commonly accompany 'Cancer'?",
"cypher": "MATCH (d1:Disease {{name:'Cancer'}}) -[:acompany_with]-> (d2:Disease) RETURN
    d2.name",

Now it's your turn to generate the question and Cypher pairs:
```

Figure 11: The prompt used to generate questions.

Ent. Type	# Ent.	Examples
Check	3,353	Bronchography
Department	54	Department of Plastic and Reconstructive Surgery
Disease	8,807	Thrombosed Vasculitis
Drug	3,828	Jingwanhong Hemorrhoid Cream
Food	4,870	Tomato and Vegetable Beef Ball Soup
Producer	17,201	Tongyi Pharmaceutical Penicillin V Potassium Tablets
Symptom	5,998	Hypertrophy of breast tissue
Total	**44,111**	/

Table 3: Entities in LHY Database.

Rel. Type	# Rel.	Examples
belongs_to	8,844	<Gynaecology, belongs_to, Obstetrics and Gynaecology>
common _drug	14,649	<Yang Qiang, common_drug, Phentolamine mesylate dispersible tablets>
do_eat	22,238	<Thoracic spine fracture, do_eat, Blackfish>
drugs_of	17,315	<Penicillin V Potassium Tablets, drugs_of, Tongyi Pharmaceuticals Penicillin V potassium tablets>
need _check	39,422	< Unilateral emphysema, need_check, Bronchography>
no_eat	22,247	<Lip disease, no_eat, Almonds>
recommend_drug	59,467	<Mixed hemorrhoids, recommend_drug, Jingwanhong Hemorrhoid Cream>
recommend_eat	40,221	<Synovial effusion, recommend_eat, Beef Ball Soup with Tomato and Vegetable Punch>
has_ symptom	5,998	<Early Breast Cancer, has_symptom, Hypertrophy of breast tissue>
accompany _with	12,029	<Valvular insufficiency of the traffic veins of the lower extremities, accompany_with, Thromboembolic vasculitis>
Total	**294,149**	/

Table 4: Relations in LHY Database.

Ent. Type	# Ent.	Examples
Anatomy	402	Digestive System
Biological_process	11,381	Protein Sialylation
Cellular_component	1,391	Meiotic Spindle
Compound	1,552	Mannitol
Disease	137	Hypertension
Gene	20,945	STRIP2
Molecular_function	2,884	Vitamin Transporter Activity
Pathway	1,822	Glycolysis
Pharmacologic_class	345	Decreased Blood Pressure
Side_effect	5,734	Subileus
Symptom	438	Ageusia
Total	**47,031**	/

Table 5: Entities in Hetionet Database.

Rel. Type	# Rel.	Examples
Anatomy–downregulates–Gene	102,240	<Bronchus, downregulates, GRIA2>
Anatomy–expresses–Gene	526,407	<Myocardium, expresses, EFHD1>
Anatomy–upregulates–Gene	97,848	<Adipose tissue, upregulates, PARM1>
Compound–binds–Gene	11,571	<Sildenafil, binds, CYP3A4>
Compound–causes–Side_Effect	138,944	<Ciprofloxacin, causes, Visual Disturbance>
Compound–downregulates–Gene	21,102	<Tacrolimus, downregulates, UBE2C>
Compound–palliates–Disease	390	<Fluvoxamine, palliates, Panic Disorder>
Compound–resembles–Compound	6,486	<Clotrimazole, resembles, Bifonazole>
Compound–treats–Disease	755	<Reserpine, treats, Hypertension>
Compound–upregulates–Gene	18,756	<Estriol, upregulates, KLHL9>
Disease–associates–Gene	12,623	<Parkinson's Disease, associates, HTR7>
Disease–downregulates–Gene	7,623	<Schizophrenia, downregulates, MLST8>
Disease–localizes–Anatomy	3,602	<Migraine, localizes, Brain>
Disease–presents–Symptom	3,357	<Lung Cancer, presents, Constipation>
Disease–resembles–Disease	543	<Bone Cancer, resembles, Head and Neck Cancer>
Disease–upregulates–Gene	7,731	<Malaria, upregulates, JAK2>
Gene–covaries–Gene	61,690	<IMP3, covaries, OR8U8>
Gene–interacts–Gene	147,164	<TRIM27, interacts, MED21>
Gene–participates–Biological_Process	559,504	<ABCA1, participates, Lipid Homeostasis>
Gene–participates–Cellular_Component	73,566	<KLHL14, participates, Neuronal Cell Body>
Gene–participates–Molecular_Function	97,222	<TOP2B, participates, ATPase Activity>
Gene–participates–Pathway	84,372	<GGT5, participates, Metabolism>
Gene-regulates-Gene	265,672	<BCCIP, regulates, HLTF>
Pharmacologic_Class–includes–Compound	1,029	<Allergens, includes, Benzocaine>
Total	**2,250,197**	/

Table 6: Relations in Hetionet Database.

LLM name	LLM version	LLM site
GPT	gpt-3.5-turbo-16k	https://platform.openai.com/docs/models/gpt-3.5-turbo
Llama3	Meta-Llama-3-8B	https://huggingface.co/meta-llama/Meta-Llama-3-8B-Instruct
InternLM2	internlm2-7B	https://huggingface.co/internlm/internlm2-base-7b
Qwen2	Qwen2-7B	https://huggingface.co/Qwen/Qwen2-7B

Table 7: Versions of the backbone LLMs

Database	Pipeline	Grammatical Validator	Semantic Validator	Entity Validator	Schema Validator	Coherence Validator	All passed
LHY	LLM-based prompting	99.69%	N/A	99.62%	82.77%	N/A	83.87%
LHY	Template-filling	99.87%	92.34%	100%	99.87%	28.59%	27.21%
Hetionet	LLM-based prompting	96.08%	N/A	99.08%	61.69%	N/A	64.79%
Hetionet	Template-filling	100%	91.81%	99.52%	100%	38.15%	36.66%

Table 8: MedT2C's passing rates of each automatic validator.

```
You are an experienced Cypher developer, English Master, and a helpful assistant that helps
    me to verify whether the cypher is coherent with the question!
You will be given a JSON object containing a question and a cypher query. You should first
    take a look at the schema provided below. The schema is the graph database on which the
    cypher queries will be run.

The schema:

{schema}

You must organize your answer step by step and in the end, you should make your judgment.

Here are three areas that you should pay attention to:
1. whether the output of cypher is coherent with the question, which means that the output of
    cypher must contain the information that the question asks.
2. If the question points out a piece of key information, you should check whether this key
    information is pointed out in the cypher. For example, if the question provides a piece
    of exact information such as the exact name of the disease, this information can not be
    inconsistent in the cypher. If there is no exact key information, you can skip this area.
3. whether this cypher answers the question provided in the JSON object. You should simulate
    the cypher step by step according to the schema provided. Then you should judge whether
    this cypher is in line with the question.

You should make your judgment according to these three areas. If there are no problems in
    these three areas in the cypher, you must answer with 'True'. Otherwise, you should
    answer with 'False'.

Here are some example JSON objects:

{example}

Now it's your turn to answer! Here is the JSON object you should evaluate:

{json_object}

Now evaluate carefully the JSON object and provide your answer step by step.
```

Figure 12: The prompt used in Semantic validator.

```
<|Example 1|>
{
    "question": "Which diseases belong to the 'Psychiatry' department?",
    "cypher": "MATCH (d:Disease)-[:belongs_to]->(dept:Department) WHERE dept.name =
        'Neurology' RETURN d.name"
},
<|Answer 1|>
The cypher is not in line with the question because the question is to find the diseases in
    the 'Psychiatry' department but the department name in the cypher is 'Neurology'
    department.
Since the key information is inconsistent, I would mark this JSON object as False.

<|Example 2|>
{
    "question": "Which foods should be avoided for the disease 'Brain tumor'?",
    "cypher": "MATCH (d1:Disease {name:'Brain tumor'})-[:no_eat]->(d2:Food) RETURN d2.name"
}
<|Answer 2|>
Firstly, the output of cypher contains the key information 'the food' asked by the question.
Secondly, the key information 'Brain tumor' provided in the question is contained in the
    cypher.
Finally, the logic of cypher is exactly similar to the question.
So, I think this JSON object is True.

<|Example 3|>
{
    "question": "What pathways do the genes 'BRCA1' and 'BRCA2' participate in?",
    "cypher": "MATCH (g:Gene)-[:PARTICIPATES_GpPW]->(:Pathway) WHERE g.name IN ['BRCA1',
        'BRCA2'] RETURN g.name"
}
<|Answer 3|>
There are two errors.
Firstly, as the question asks for pathways but the output of cypher is the name of the gene,
    the output of the cypher is inconsistent with the question.
Secondly, the question is to find the pathway that both the genes 'BRCA1' and 'BRCA2'
    participate in. But the cypher matches the pathways that 'BRCA1' or 'BRCA2' participates
    in. The logic 'AND' and 'OR' are totally different.
Therefore, I think this JSON object is False.
```

Figure 13: The English few-shot examples used in the Semantic Validator.

```
You are an experienced medical assistant who has mastered English and medical knowledge.

You will be given a question and the responses given by the doctor. The doctor is very
    professional, he gives direct responses. But he sometimes misunderstands the problem.
    Your task is to check if the results are coherent with the question by analyzing the
    category.
For example, if the question asks for food and the answer is food, in this case, it is
    relevant because the category is the same. Even if the foods don't seem to be directly
    related, you can not deny them because the doctor is professional.
But if the question asks for food, the doctor gives the response on sports. You should point
    out this error because the category is different.

As a medical assistant, you just need to pay attention to whether the category of the answer
    corresponds to the category that the question asks. You don't need to think about the
    reasonableness of the answer.
Answer with 'True' if the category is the same. Otherwise, answer with 'False'.
You need to carefully explain your answer.

Here are some examples of questions and results:

<Example 1>
Question: Find out the diseases associated with the 'Oncology' department.
Responses by the doctor: Breast cancer, Pancreatic cancer, Colon cancer
Your reply: Breast cancer, pancreatic cancer, and colon cancer belong to the Oncology
    department. And the question asks for diseases. So I think it is relevant, and my answer
    is True.

<Example 2>
Question: Which foods should be avoided for the disease 'Coeliac disease'?
Responses by the doctor: Swimming, Running, Biking, Walking
Your reply: The responses are sports. But this question asks for food. So I think it is not
    relevant, my answer is False.

Now it's your turn to verify if the responses are relevant to the question.
Remember! You just need to pay attention to whether the answer corresponds to the question.
    You don't need to think about the reasonableness of the answer.

Question:{question}
Responses by the doctor: {results}
Your reply:
```

Figure 14: The prompt used in Coherence Validator.

```
You are an expert in medical field and Cypher query language. You are asked to evaluate the
     quality of the Cypher queries generated by 2 models for the same question. You will be
     first given the question written in natural language. Then you will be given the Cypher
     queries generated by 2 models. Your task is to compare the quality of these two Cyphers
     and select the better one. You should consider the following aspects when selecting the
     better Cypher:
    1. Syntactical correctness: whether the Cypher query is syntactically correct;
    2. Semantic correctness: whether the Cypher query can correctly answer the question;
    3. Readability: whether the Cypher query is easy to read and understand;
    4. Efficiency: whether the Cypher query is efficient in terms of time and space
         complexity;
    5. Conciseness: whether the Cypher query is concise and clear;
    6. Completeness: whether the Cypher query can cover all the necessary information in the
         database.
    You should select the better Cypher query based on these aspects. Output your selected
         Cypher as well as your reasons.

Here is the question:
{
    "question": "{{ question }}"
}

Here are the outputs of the models:
[
    {
        "number": "1",
        "cypher": "{{ cypher_1 }}"
    },
    {
        "number": "2",
        "cypher": "{{ cypher_2 }}"
    }
]
Your output should be in the following format, DO NOT output anything other than this JSON
     object:
{
"better_cypher": "1",
"reason": "reasons why 1 is selected"
}

Now select the better Cypher and give your reasons:
```

Figure 15: The prompt used in Cypher quality evaluation.