Text2Cypher: Bridging Natural Language and Graph Databases

Makbule Gulcin Ozsoy
Neo4j / London, UK
makbule.ozsoy@neo4j.com

Leila Messallem
Neo4j / Malmö, Sweden
leila.messallem@neo4j.com

Jon Besga
Neo4j / London, UK
jon.besga@neo4j.com

Gianandrea Minneci
Neo4j / London, UK
gianandrea.minneci@neo4j.com

Abstract

Knowledge graphs use nodes, relationships, and properties to represent arbitrarily complex data. When stored in a graph database, the Cypher query language enables efficient modeling and querying of knowledge graphs. However, using Cypher requires specialized knowledge, which can present a challenge for non-expert users. Our work Text2Cypher aims to bridge this gap by translating natural language queries into Cypher query language and extending the utility of knowledge graphs to non-technical expert users. While large language models (LLMs) can be used for this purpose, they often struggle to capture complex nuances, resulting in incomplete or incorrect outputs. Fine-tuning LLMs on domain-specific datasets has proven to be a more promising approach, but the limited availability of high-quality, publicly available Text2Cypher datasets makes this challenging. In this work, we show how we combined, cleaned and organized several publicly available datasets into a total of 44,387 instances, enabling effective fine-tuning and evaluation. Models fine-tuned on this dataset showed significant performance gains, with improvements in Google-BLEU and Exact Match scores over baseline models, highlighting the importance of high-quality datasets and fine-tuning in improving Text2Cypher performance.

1 Introduction

Databases are essential in applications, supporting data storage and knowledge management, and are typically accessed via query languages like SQL (for relational databases) or Cypher (for graph databases). With advancements in LLMs, users can now query databases using natural language through applications that perform tasks such as Text2SQL or Text2Cypher. Consequently, even with minimal technical expertise, users can easily retrieve information, build applications such as dashboards or analytics, or integrate knowledge into other systems, such as Retrieval-Augmented Generation (RAG). The Text2Cypher task converts plain language questions into Cypher query language (see Figure 1). In the figure, a user wants to write a Cypher query for *'What are the movies of Tom Hanks'*. A Text2Cypher model translates the input natural language question into Cypher, i.e., it returns *'MATCH (actor:Person {name: "Tom Hanks"})-[:ACTED_IN]->(movie:Movie) RETURN movie.title AS movies'*. This generated Cypher query can then be used to retrieve relevant data from the database, allowing for utilization based on the needs of the user.

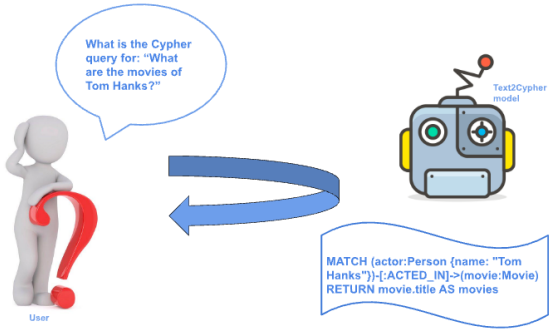

Figure 1: User wants to write a Cypher query for 'What are the movies of Tom Hanks'. A Text2Cypher model translates the input natural language question into Cypher, i.e., 'MATCH (actor:Person {name: "Tom Hanks"})-[:ACTED_IN]->(movie:Movie) RETURN movie.title AS movies'

Foundational large language models (LLMs) can be utilized for Text2Cypher task directly with an appropriate prompt. However, they may struggle with complex queries, leading to incomplete or incorrect outputs which damage the utility of the knowledge graph. Fine-tuning LLMs on domain-specific datasets offers a promising solution but requires high quality data that pairs natural language queries with Cypher translations, plus schema information for greater accuracy. However, creating such a dataset is challenging, as it requires an understand-

Proceedings of the Generative AI and Knowledge Graph Workshop (GenAIK), pages 100–108

ing of graph representation, domain-specific knowledge to formulate effective natural language questions, and proficiency in Cypher syntax. If the training set does not include high-quality, diverse and sufficient examples, the fine-tuned Text2Cypher model may underperform.

The number of publicly available Text2Cypher datasets is limited. A few examples include those created by Neo4jLabs (Neo4jLabs, 2024), datasets converted from Text2SQL sets (Zhao et al., 2023c,b; SemanticParser4Graph, 2024), and others constructed synthetically (Zhong et al., 2024). However, these datasets are prepared independently, which makes it difficult to use them together. In this work, we combine and refine instances from publicly available datasets, creating a large dataset for training and testing, and use it to benchmark and fine-tune foundational models for Text2Cypher.

Our main contributions are as follows:

- We combine instances from publicly available datasets, refining and organizing them to enhance usability. The final dataset includes $44,387$ instances, with a training and test split, of $39,554$ and $4,833$ instances, respectively. The dataset is made available to the public [1].
- We use this new dataset to benchmark foundational and previously fine-tuned models on the Text2Cypher task. The results showed that large-foundational models performed the best, however, the fine-tuned models showed promise for improving performance.
- We fine-tuned a set of selected foundational models using the new dataset and compared their performance to benchmark results. The results showed that all the fine-tuned models achieve better results than their baseline models. One of the fine-tuned models are made publicly accessible [2].

The paper is structured as follows: Section 2 discusses related work on translating natural language to query languages, with a focus on Text2Cypher. Section 3 details the dataset preparation process. Section 4 and Section 5 present our experiments for benchmarking and fine-tuning. Finally, Section 6 concludes the paper.

[1]Dataset: https://huggingface.co/datasets/neo4j/text2cypher-2024v1

[2]A finetuned model: https://huggingface.co/neo4j/text2cypher-gemma-2-9b-it-finetuned-2024v1

2 Related Work

2.1 Graph Databases and Cypher Language

Graph Database Systems store, manage, and retrieve graph data, where nodes, relationships, and their properties are used for representing real-world knowledge (Zheng et al., 2024). These systems enable efficient querying of relationships and offer easy visualization (Yoon et al., 2017).

Companies specializing in graph databases include Neo4j (Neo4j, 2024), NebulaGraph (Wu et al., 2022), and Amazon Neptune (Bebee et al., 2018). In April 2024, GQL standard (ISO/IEC 39075:2024) (languages – GQL, 2024) was released, providing a unified query language for graph databases. The ISO GQL standard is heavily influenced by Neo4j's Cypher language (both share a large amount of syntax and they are both declarative pattern-matching languages). So while this work focuses on translating natural language into Cypher queries, the general approach will be applicable to GQL when it is more widespread.

2.2 Natural Language to Code Generation

Converting natural language to executable code is essential for applications like database interfaces and virtual assistants (Pasupat and Liang, 2015; Yu et al., 2018; Agashe et al., 2019; Lai et al., 2023; Zhong et al., 2024). Advancements in large language models (LLMs) have enabled significant progress in translating natural language into query languages like SQL or Cypher. This capability allows users to retrieve information, build dashboards, and integrate database knowledge into systems like Retrieval-Augmented Generation (RAG).

There has been extensive research on the Text2SQL task, which translates natural language queries to SQL (Yu et al., 2018; Guo et al., 2019; Rajkumar et al., 2022; Li et al., 2023; Fan et al., 2024; Li et al., 2024). In contrast, there is less work focused on the Text2Cypher task, which translates natural language queries into Cypher. This disparity stems from SQL's dominance in relational databases and traditionally high industry demand (Memgraph, 2024). However, graph-based data representation is not only a more obvious fit for knowledge graphs, but is gaining recognition for addressing issues like hallucinations in RAG models. As such interest in Cypher is increasing, and Cypher's efficiency in expressing complex, interconnected queries makes it a compelling alternative to SQL for knowledge graphs (and other domains).

2.3 Text2Cypher Task

The Text2Cypher task translates natural language queries into Cypher queries (see Figure 1). Large language models (LLMs) can handle this with zero- or few-shot prompts, which have shown promise but are still imperfect (Chen et al., 2021). Fine-tuning LLMs offers a more robust alternative, though it is limited by the scarcity of relevant datasets and high computational costs (Ni et al., 2023). Some research has focused on creating datasets for Text2Cypher, while others have concentrated on model benchmarking and fine-tuning for this task.

Some dataset preparation efforts for Text2Cypher involve translating existing datasets from other query languages, while others focus on creating dedicated datasets. Examples of translations include S2CTrans (Zhao et al., 2023a), which converts SPARQL queries into Cypher in order to handle complex graph queries, and CySpider (Zhao et al., 2023b) and Rel2Graph (Zhao et al., 2023c), which map SQL queries to Cypher and create parallel corpora of natural language-to-Cypher pairs. Specific Text2Cypher datasets include Neo4jLabs datasets (Neo4jLabs, 2024), which are generated via LLMs and their crowd-sourcing tool (Bratanič, 2024c). Opitz and Hochgeschwender (Opitz and Hochgeschwender, 2022) and SyntheT2C (Zhong et al., 2024) used synthetic methods to generate Cypher query data. While several efforts have been made to create datasets for the Text2Cypher task, these datasets are often developed independently. In this work, we aim to compile a well-structured Text2Cypher dataset by combining and structuring instances from publicly available sources.

Some research has focused on benchmarking and fine-tuning models for the Text2Cypher task: Authors from Neo4j (Bratanič, 2024a) released fine-tuned models based on their datasets, using LLMs like LLama and Codestral. GPT4Graph (Guo et al., 2023) evaluated LLMs on graph tasks, including Cypher query generation, using the MetaQA (Zhang et al., 2018) dataset and testing InstructGPT-3 (Ouyang et al., 2022) in zero- and one-shot settings. TopoChat (Xu et al., 2024) developed a material sciences dataset, using prompts to generate Cypher queries with foundational LLMs. Baraki et al. (Baraki, 2024) leveraged Neo4jLabs' crowd-sourced and synthetic datasets to fine-tune models, using the crowd-sourced set for evaluation. TransKGQA (Chong et al., 2024) extracted information from knowledge graphs, using the 'sentence-transformers/all-MiniLM-L12-v2' model to generate Cypher queries. Although these works have provided fine-tuned models, the number of models used was limited. In our work, after constructing a larger and more organized dataset, we benchmark and fine-tune a wider range of baseline LLMs.

Table 1: Data fields

Field name	Description
question	Textual question
schema	The database schema
cypher	Output cypher query
data_source	Alias of the dataset source
database_reference	Alias of the database
instance_id	Incremental index

3 Dataset Construction

While several Text2Cypher datasets exist, many are prepared separately, making them hard to use together. In this work we bring instances from publicly available datasets together, clean and organize them for smoother use. For this purpose, we executed three main steps: (i) Identification and collection of publicly available datasets, (ii) Combining and cleaning the data, and (iii) Creating the training and test splits.

3.1 Identification and collection of publicly available datasets

As the initial step, we identified the datasets which are already publicly available. We have identified 25 different resources from (i) Neo4j resources (including Neo4jLabs) (ii) HuggingFace (HF) datasets and (iii) Academic papers. Out of these resources, we were able to utilize 16 of those datasets, as they met our criteria of including natural language question and Cypher query pairs, as well as database schema information, along with appropriate licensing and accessibility.

3.2 Combining and cleaning the data

After identifying the input datasets, we standardized them into a single format. Each row was reformatted to include fields ["question", "schema", "cypher", "data_source", "database_reference", "instance_id"], as described in Table 1. One of the fields, namely "database_reference", requires particular attention. In some cases within the com-

bined dataset, database access is available where the reference or the generated Cypher queries can be executed. Further details about these databases can be found at the page of Neo4jLabs-Crowdsourcing Initiative (Bratanič, 2024c). The combined dataset is further cleaned in two steps:

- **Manual checks and updates**: This step aims to produce more reliable and error-free output data. Queries are manually reviewed, and errors are corrected through straightforward removals or updates: (i) Updating Cypher queries, such as removing unwanted characters (e.g., back-tick) (ii) Removing irrelevant questions (e.g., "Lorem ipsum . . . ") (iii) Deduplicating rows based on the ["question", "cypher"] pairs.

- **Syntax validation**: Each Cypher query is checked for syntax errors by running 'EXPLAIN' clauses in a local Neo4j database. Queries that trigger syntax errors are identified and removed from the combined dataset. Additionally, the queries are de-duplicated.

3.3 Creating the training and test splits

Having the cleaned dataset, the final step is to prepare the training and test splits. We have identified 3 groups of datasets: (i) Train-specific datasets: Files with "train" in the name, used for training. (ii) Test-specific datasets: Files with "test" or "dev" in the name, used for testing. (iii) Remaining datasets: Files with no specified use. We assigned Train-specific datasets to the training split and Test-specific datasets to the test split. The remaining datasets were split 90:10 for training and testing, respectively. Each split was shuffled to prevent over-fitting from sequence or repetitive questions.

The data preparation resulted in $44,387$ instances, with $39,554$ instances in the training split and $4,833$ instances in the test split. The train and test splits contain $\sim$89% and $\sim$11% of the overall data, respectively. Their distribution across data sources is similar, as shown in Figure 2. As explained previously, not every instance in the training and test sets has database access, as indicated by the "database_reference" field. Analyzing the distribution of instances with database access reveals that the training set contains $22,093$ such instances (55.85% of the total), while the test set has $2,471$ instances (51.12% of the total). These instances are later used in the experimentation with an additional evaluation procedure.

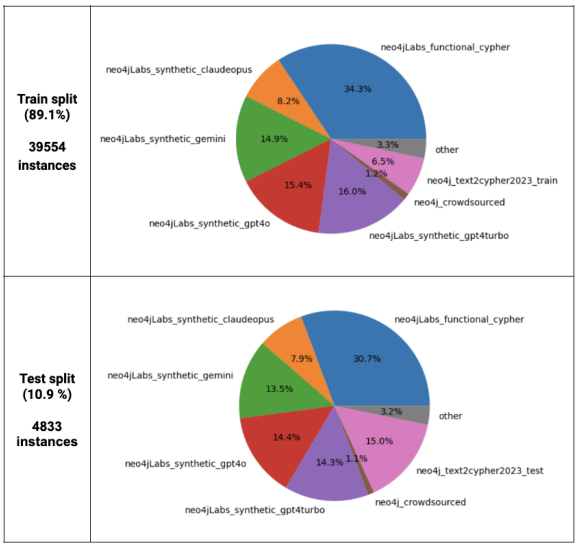

Figure 2: Data distribution: The train and test splits contain $\sim$89% and $\sim$11% of the overall data, respectively.

4 Model Evaluation and Benchmarking

After constructing a larger and more organized dataset, this section presents the benchmarking results.

4.1 Evaluation metrics

Text2Cypher is a type of text-to-text generation task, where natural language questions are translated into Cypher queries. Therefore, evaluation metrics commonly used in other text-to-text tasks, such as machine translation and summarization, can also be applied to this task. Using HuggingFace Evaluate library (HuggingFace, 2024), we computed: (i) Text2Text comparison metrics, such as ROUGE, BLEU, METEOR (ii) Embedding similarity metrics, such as BERTScore, FrugalScore (iii) Text similarity metrics, such as Cosine and Jaro-Winkler similarity, and (iv) Exact Match score. Although we calculated all these metrics, we primarily use Google-BLEU and Exact Match scores throughout the paper.

4.2 Experimental Setup

For benchmarking the models, we used the test split of the larger dataset introduced in Section 3. Closed models were evaluated through APIs provided by the respective companies. For the other models, which are openly accessible via HuggingFace (HF), we utilized HF interfaces. To access GPUs for evaluation, we employed RunPod (RunPod, 2024) environments. Where relevant, we followed the instructions outlined in Table 3, which

Table 2: Models used for benchmarking

Type	Name	Base model
HF	hf_ft_lakkeo_stable_cypher_instruct3B	Stability AI/Stable-code-instruct-3b
HF	hf_ft_tomasonjo_text2cypher	Meta/Llama-3-8b-Instruct
HF	hf_ft_neo4j_text2cypher_23_codellama	Meta/CodeLlama13B
OpenAI	openai_ft_neo4j_text2cypher_23_gpt3_5	OpenAI/GPT3.5
HF	hf_foundational_meta_llama3_1_8B_instruct	Meta/LLama-3.1-8B-instruct
HF	hf_foundational_codeLlama_7B_instruct_hf	Meta/CodeLLama-7B-instruct
HF	hf_foundational_gemma2_9B_it	Google/Gemma-2-9B-it
HF	hf_foundational_codegemma_7B_it	Google/CodeGemma-7B-it
OpenAI	openai_gpt3_5	OpenAI/GPT-3.5
OpenAI	openai_gpt4_o	OpenAI/GPT-4o
OpenAI	openai_gpt4_o_mini	OpenAI/GPT-4o-mini
VertexAI	gemini-1.0-pro-002	Google/Gemini-1.0-Pro
GoogleAIStudio	gemini-1.5-flash-001	Google/Gemini-1.5-Flash
GoogleAIStudio	gemini-1.5-pro-001	Google/Gemini-1.5-Pro

Table 3: Instructions used

Type	Instruction prompt
System Instruct.	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 in the schema. 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.
User Instruct.	Generate Cypher statement to query a graph database. Use only the provided relationship types and properties in the schema. Schema: {schema} Question: {question} Cypher output:

were inspired from tips provided by authors from Neo4j (Bratanič, 2024b).

We defined two types of evaluation procedures:

- **Translation-based evaluation**: The generated Cypher queries are compared with the reference Cypher queries based solely on the textual content. The evaluation metrics used for this comparison are detailed in Section 4.1.
- **Execution-based evaluation**: The generated and reference Cypher queries are executed on the target databases, and their outputs are collected. The collected execution results are converted into string representations (ordered lexicographically for consistency). The same evaluation metrics used in the translation-based evaluation are then applied to these outputs.

4.3 Benchmarking results

For benchmarking, we aimed to evaluate not only baseline LLMs but also previously fine-tuned models specifically tailored for the Text2Cypher task. The list of models used for benchmarking purpose are listed in Table 2. In the table, first group includes the fine-tuned models, second group includes the open-weighted models and the last group includes the closed models.

Figure 3 presents the performance comparison of the selected models on the test split. The figure presents Google-BLEU score for translation-based and Exact Match score for execution-based evaluation. Among the previously fine-tuned models, i.e., with different data, HF/tomasonjo_text2cypher performed best, but this may be misleading as it had encountered 14.4% of the test data during training. Among the open-weighted models, Google/Gemma-2-9B-it is the best performing model. Contrary to expectations, the code-focused models (e.g., CodeGemma) did not outperform the baseline models. This may be attributed to the fact that Cypher queries are relatively closer to natural language, reducing the advantage of code-specific models. Among closed-

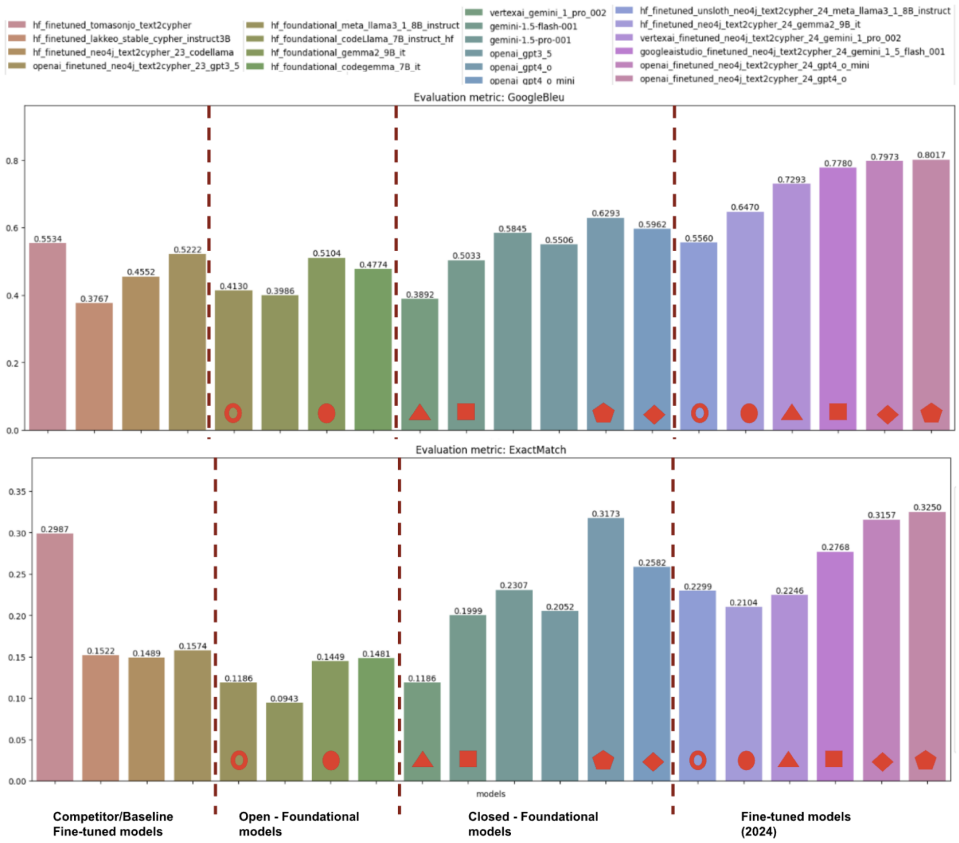

Figure 3: Performance comparison of the baseline and finetuned models. Presents Google-BLEU score for translation-based and Exact Match score for execution-based evaluation.

foundational models, the best performing models are OpenAI/GPT4o, OpenAI/GPT4o-mini, and Google/Gemini-1.5-Pro-001 led in performance, with larger models outperforming smaller ones.

Overall, closed foundational models like GPT and Gemini achieved the best performance, though at higher costs. Fine-tuned models improved baseline open-weighted models. In the next section, we explore the process of fine-tuning models and evaluating them using the new dataset introduced in Section 3.

5 Model Finetuning and Evaluation

Based on the findings of benchmarking, presented in Section 4, we selected six baseline models for our subsequent steps, presented in Table 4. In the table, first group includes the open-weighted models, while the second group includes the closed models. Although some models, such as Google/Gemini-1.5-Pro, demonstrated better performance in the benchmark results, they were unavailable for fine-tuning at the time of this analysis and are therefore not included in this work.

Table 4: Models used for fine-tuning

Type	Base model
HF	Meta/LLama-3.1-8B-instruct
HF	Google/Gemma-2-9B-it
OpenAI	OpenAI/GPT-4o
OpenAI	OpenAI/GPT-4o-mini
VertexAI	Google/Gemini-1.0-Pro
GoogleAIStudio	Google/Gemini-1.5-Flash

5.1 Experimental setup

For the finetuning process, we used the training split of the larger dataset introduced in Section 3. The closed models were trained through APIs provided by their respective companies, while the other models were finetuned using HuggingFace (HF) or Unsloth (Unsloth, 2024) interfaces on GPU machines hosted in RunPod (RunPod, 2024) environments. The evaluation procedures and metrics were identical to those used in benchmarking section, Section 4. The instructions remained consistent with those outlined in Table 3. We used Google-BLEU score for translation-based and Exact Match score for execution-based evaluation.

Table 5: The improvements of the fine-tuned models over the baseline models

Baseline model	ΔGoogle BLEU	ΔExact Match
HF/LLama3.1-8B-it	∼0.14	∼0.11
HF/Gemma2-9B-it	∼0.13	∼0.07
VertexAI/Gemini-1.0-Pro-002	∼0.34	∼0.11
GoogleAIStudio/Gemini-1.5-Flash-001	∼0.27	∼0.09
OpenAI/Gpt-4o-mini	∼0.20	∼0.06
OpenAI/Gpt-4o	∼0.18	∼0.01

5.2 Finetuning results

The evaluation results for all models, including those previously benchmarked, are shown in Figure 3. The last group in the figure highlights the fine-tuned models trained on the dataset introduced in Section 3. For easier comparison, red shapes are used to link each fine-tuned model with its corresponding baseline version. The figure shows that all fine-tuned models achieve better results than their baseline models. The best results are obtained by the Finetuned-OpenAI/Gpt4o, Finetuned-OpenAI/Gpt4o-mini and Finetuned-GoogleAIStudio/Gemini-1.5-Flash-001 models.

The improvements in the fine-tuned models over the baseline models are presented in Table 5. The enhancements for models that have already performed well are relatively smaller than others. For example, OpenAI/Gpt-4 shows an 0.18 increase in the Google-BLEU score, while VertexAI/Gemini-1.0-Pro-002 demonstrates a 0.34 increase. The improvements of the finetuned open-weighted models, i.e. HF/LLama3.1-8B-it and HF/Gemma2-9B-it, are relatively less pronounced. During fine-tuning of these models, our goal was to minimize resource usage (e.g., cost and memory). As a result, with better-tuned parameters, we could potentially achieve even stronger results.

Although all the fine-tuned models showed improvements in Google-BLEU and Exact Match scores, it is important to remain aware of the potential risks and pitfalls associated with fine-tuning.

6 Conclusion

Databases are essential for data storage, management, and retrieval, often accessed through query languages like Cypher. Recent advancements in large language models (LLMs) have made it possible to access databases using natural language through tasks like Text2Cypher. While LLMs can be directly used for this task, they often struggle with complex queries, resulting in incomplete or incorrect Cypher outputs. Fine-tuning LLMs on specific Text2Cypher datasets offers a more effective solution. However, publicly available Text2Cypher datasets are limited and often created independently, making them difficult to combine and use effectively. To address this, we combined and refined several datasets into a unified set of 44,387 instances, with 89% in the training split and 11% in testing. Fine-tuned models trained on this dataset outperformed baselines, achieving up to a 0.34 increase in Google-BLEU score and a 0.11 increase in Exact Match score. This work highlights the importance of dataset and fine-tuning for Text2Cypher task. Future work will refine the dataset further, analyze challenging cases, and explore improvements through prompt engineering and model optimization.

Limitations

The previous sections demonstrated how fine-tuned models significantly boost performance. However, there are several risks and pitfalls that must be considered.

Even though we de-duplicated the dataset by ["question", "cypher"] pairs, it is still possible to have instances where the same "question" appears with different "cypher" outputs. In such cases, these instances may have been split between the training and test sets, meaning that fine-tuned models could have already encountered the same "question" during training. However, since these instances have different "cypher" outputs, even if the fine-tuned models memorize the "cypher" output for the question, their generated response would be incorrect. This essentially penalizes the models for having seen and memorized the question. In the future, we plan to clean the test set of such instances, re-run the evaluation, and assess any performance differences.

Our dataset is constructed by collecting and combining publicly available datasets, which may include paraphrased versions of the same questions. It is known that training on paraphrased examples of the test set may artificially inflate the performance of the fine-tuned model (Yang et al., 2023). Additionally, both the training and test sets are drawn from the same data distribution, sampled

from a larger dataset. If the data distribution shifts, the results may not hold up in the same way.

Finally, the dataset used was gathered from publicly available sources. Over time, foundational models may gain access to both the training and test sets, potentially achieving similar or even better performance results in the future.

References

Rajas Agashe, Srinivasan Iyer, and Luke Zettlemoyer. 2019. Juice: A large scale distantly supervised dataset for open domain context-based code generation. *arXiv preprint arXiv:1910.02216*.

Welemhret Welay Baraki. 2024. Leveraging large language models for accurate cypher query generation: Natural language query to cypher statements. Master degree project, University of Skövde. https://www.diva-portal.org/smash/get/diva2:1881385/FULLTEXT01.pdf.

Bradley R Bebee, Daniel Choi, Ankit Gupta, Andi Gutmans, Ankesh Khandelwal, Yigit Kiran, Sainath Mallidi, Bruce McGaughy, Mike Personick, Karthik Rajan, et al. 2018. Amazon neptune: Graph data management in the cloud. In *ISWC (P&D/Industry/BlueSky)*.

Tomaž Bratanič. 2024a. Hf models. https://huggingface.co/tomasonjo.

Tomaž Bratanič. 2024b. Langchain cypher search: Tips and tricks. https://neo4j.com/developer-blog/langchain-cypher-search-tips-tricks/.

Tomaž Bratanič. 2024c. Neo4jlabs crowdsourcing initiative. https://medium.com/@bratanic-tomaz/e65ba51916d4.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

You Li Chong, Chin Poo Lee, Shahrin Zen Muhd-Yassin, Kian Ming Lim, and Ahmad Kamsani Samingan. 2024. Transkgqa: Enhanced knowledge graph question answering with sentence transformers. *IEEE Access*.

Yuankai Fan, Zhenying He, Tonghui Ren, Can Huang, Yinan Jing, Kai Zhang, and X Sean Wang. 2024. Metasql: A generate-then-rank framework for natural language to sql translation. *arXiv preprint arXiv:2402.17144*.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

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. *arXiv preprint arXiv:2305.15066*.

HuggingFace. 2024. Huggingface evaluate. https://huggingface.co/evaluate-metric.

Yuhang Lai, Chengxi Li, Yiming Wang, Tianyi Zhang, Ruiqi Zhong, Luke Zettlemoyer, Wen-tau Yih, Daniel Fried, Sida Wang, and Tao Yu. 2023. Ds-1000: A natural and reliable benchmark for data science code generation. In *International Conference on Machine Learning*, pages 18319–18345. PMLR.

ISO/IEC 39075:2024 Information Technology – Database languages – GQL. 2024. Gql database languages. https://jtc1info.org/slug/gql-database-language/.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36.

Memgraph. 2024. Graph database vs relational database. https://memgraph.com/blog/graph-database-vs-relational-database.

Neo4j. 2024. Neo4j graph database. https://neo4j.com/.

Neo4jLabs. 2024. Neo4jlabs datasets. https://github.com/neo4j-labs/text2cypher/tree/main/datasets.

Ansong Ni, Srini Iyer, Dragomir Radev, Veselin Stoyanov, Wen-tau Yih, Sida Wang, and Xi Victoria Lin. 2023. Lever: Learning to verify language-to-code generation with execution. In *International Conference on Machine Learning*, pages 26106–26128. PMLR.

Dominik Opitz and Nico Hochgeschwender. 2022. From zero to hero: generating training data for question-to-cypher models. In *Proceedings of the 1st International Workshop on Natural Language-based Software Engineering*, pages 17–20.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

RunPod. 2024. Runpod. https://www.runpod.io/.

SemanticParser4Graph. 2024. Semanticparser4graph datasets. https://github.com/22842219/SemanticParser4Graph/tree/main/sp_data_folder.

Unsloth. 2024. Unsloth ai - open source fine-tuning for llms. https://unsloth.ai/.

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

HuangChao Xu, Baohua Zhang, Zhong Jin, Tiannian Zhu, Quansheng Wu, and Hongming Weng. 2024. Topochat: Enhancing topological materials retrieval with large language model and multi-source knowledge. *arXiv preprint arXiv:2409.13732*.

Shuo Yang, Wei-Lin Chiang, Lianmin Zheng, Joseph E Gonzalez, and Ion Stoica. 2023. Rethinking benchmark and contamination for language models with rephrased samples. *arXiv preprint arXiv:2311.04850*.

Byoung-Ha Yoon, Seon-Kyu Kim, and Seon-Young Kim. 2017. Use of graph database for the integration of heterogeneous biological data. *Genomics & informatics*, 15(1):19.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Yuyu Zhang, Hanjun Dai, Zornitsa Kozareva, Alexander J Smola, and Le Song. 2018. Variational reasoning for question answering with knowledge graph. In *AAAI*.

Zihao Zhao, Xiaodong Ge, Zhihong Shen, Chuan Hu, and Huajin Wang. 2023a. S2ctrans: Building a bridge from sparql to cypher. In *International Conference on Database and Expert Systems Applications*, pages 424–430. Springer.

Ziyu Zhao, Wei Liu, Tim French, and Michael Stewart. 2023b. Cyspider: A neural semantic parsing corpus with baseline models for property graphs. In *Australasian Joint Conference on Artificial Intelligence*, pages 120–132. Springer.

Ziyu Zhao, Wei Liu, Tim French, and Michael Stewart. 2023c. Rel2graph: Automated mapping from relational databases to a unified property knowledge graph. *arXiv preprint arXiv:2310.01080*.

Yingying Zheng, Wensheng Dou, Lei Tang, Ziyu Cui, Yu Gao, Jiansen Song, Liang Xu, Jiaxin Zhu, Wei Wang, Jun Wei, et al. 2024. Testing gremlin-based graph database systems via query disassembling. In *Proceedings of the 33rd ACM SIGSOFT International Symposium on Software Testing and Analysis*, pages 1695–1707.

Ziije Zhong, Linqing Zhong, Zhaoze Sun, Qingyun Jin, Zengchang Qin, and Xiaofan Zhang. 2024. Synthet2c: Generating synthetic data for fine-tuning large language models on the text2cypher task. *arXiv preprint arXiv:2406.10710*.