Small, Medium, and Large Language Models for Text-to-SQL

Aiko Oliveira[1,2][0009−0001−8970−0454],
Eduardo Nascimento[1][0009−0005−3391−7813],
João Pinheiro[2][0000−0002−0909−4432],
Caio Viktor S. Avila[4][0009−0002−3899−0014],
Gustavo Coelho[1][0000−0003−2951−4972], Lucas Feijó[1][0009−0006−4763−8564],
Yenier Izquierdo[1][0000−0003−0971−8572], Grettel García[1][0000−0001−9713−300X],
Luiz André P. Paes Leme[3][0000−0001−6014−7256],
Melissa Lemos[1,2][0000−0003−1723−9897], and
Marco A. Casanova[1,2][0000−0003−0765−9636]

[1] Tecgraf Institute, PUC-Rio, Rio de Janeiro, RJ, Brazil CEP 22451-900
{aikoramalho, rogerrsn, gustavocoelho, lucasfeijo, ytorres, ggarcia,
melissa}@tecgraf.puc-rio.br
[2] Department of Informatics, PUC-Rio, Rio de Janeiro, RJ, Brazil CEP 22451-900
{jpinheiro, casanova}@inf.puc-rio.br
[3] Instituto de Computação, UFF, Niterói, RJ, Brazil CEP 24210-310
lapaesleme@ic.uff.br
[4] Federal University of Ceará, UFC, Fortaleza, CE, Brazil CEP 60440-900
caioviktor@alu.ufc.br

Abstract. This paper investigates how the model size affects the ability of a Generative AI Language Model, or briefly a GLM, to support the text-to-SQL task for databases with large schemas typical of real-world applications. The paper first introduces a text-to-SQL framework that combines a prompt strategy and a Retrieval-Augmented Generation (RAG) technique, leaving as flexibilization points the GLM and the database. Then, it describes a benchmark based on an open-source database featuring a schema much larger than the schemas of most of the databases in familiar text-to-SQL benchmarks. The paper proceeds with experiments to assess the performance of the text-to-SQL framework instantiated with the benchmark database and GLMs of different sizes. The paper concludes with recommendations to help select which GLM size is appropriate for a text-to-SQL scenario, characterized by the difficulty of the expected NL questions and the data privacy requirements, among other characteristics.

Keywords: text-to-SQL · Generative AI Language Model · Retrieval-Augmented Generation · Prompt engineering · Real-World Databases.

1 Introduction

Natural language database interfaces allow users to access databases using questions formulated in natural language (NL) [2]. A strategy for constructing such

interfaces is to use a Generative AI Language Model, or briefly a GLM, to translate the NL questions into SQL queries. Such interfaces will be called *GLM-based text-to-SQL tools*.

Real-world databases, that is, databases that support real-world applications, are especially challenging for GLM-based text-to-SQL tools. Among other characteristics, their schema is typically large and complex, may not fit in the prompt area, and may lead to queries with many joins, which are difficult to synthesize. Furthermore, the data stored in such databases may have complex semantics, which must be passed to the GLM. The primary motivation of this paper is the design of GLM-based text-to-SQL tools for real-world databases.

Considering the relevance for the construction of text-to-SQL tools, a GLM may be characterized by: the *licensing mode*, where proprietary models run only on vendors' platforms versus open-source models that can be downloaded and run on a private platform; the *context length*, that is, the number of tokens accepted as input/output, which limits the database metadata and data that can be passed to the GLM in the text-to-SQL task; and the *model size* in billions of parameters, which mostly determines the hardware requirements. The *release date* of the model is also important since more recent models tend to have better performance. Smaller GLMs are attractive from the point of view of hardware requirements, but their performance for a given NL task must be assessed.

The research question addressed in this paper is then: "How does the model size affect the ability of a GLM to support the text-to-SQL task for databases with large, complex schemas?". This question is of tantamount importance for implementing GLM-based Natural Language database interfaces.

To address this question, the paper first introduces a prototype text-to-SQL framework based on a prompt strategy, SQLQueryChain[5], combined with a Retrieval-Augmented Generation (RAG) technique, leaving as flexibilization points the GLM adopted and the database. SQLQueryChain proved effective for real-world databases [5, 19], vis-a-vis the strategies at the top of the benchmark leaderboards. The RAG technique adopted injects knowledge of the database schema and the data semantics into a GLM, and led to a considerable improvement of the accuracy of text-to-SQL Prompt strategies [5]. The retrieval step of the technique incorporates an NL question decomposition strategy to improve accuracy, not addressed in [5]. An assessment of the RAG technique with question decomposition is the first contribution of the paper.

Then, the paper describes a benchmark based on the Mondial database[6], which features a relational schema much larger than the schemas of most of the databases in familiar text-to-SQL benchmarks. The Mondial database is an open-source suitable proxy for the proprietary industrial database that motivated this paper but could not be made openly available. The benchmark includes 100 NL questions, with balanced difficulty, and their translation to SQL. The benchmark also includes a synthetic dataset with 60,000 entries to help apply RAG. It should be stressed that the benchmark is not meant to train a model for the text-to-

---

[5] https://python.langchain.com/docs/use_cases/sql/prompting
[6] Available at https://github.com/dudursn/text_to_sql_chatgpt_real_world/

SQL task but rather to test the model in a challenging scenario. The Mondial benchmark, prepared for RAG, is the second contribution of the paper.

The paper proceeds with experiments to assess the performance of the text-to-SQL framework instantiated with the Mondial database and GLMs of different sizes. The experiments suggest recommendations that help select which GLM size is appropriate for the text-to-SQL task in a given context, characterized by the difficulty of the expected NL questions and the data privacy requirements, among other characteristics. The experiments and the derived recommendations are the third contribution of the paper.

The paper is organized as follows. Section 2 summarizes the required background concepts and related work. Section 3 presents the prototype NL database interface framework and the RAG technique. Section 4 describes the Mondial benchmark. Section 5 covers the experiments and derived recommendations. Section 6 contains the concluding remarks.

2   Background Concepts and Related Work

2.1   Generative AI Language Models

The approaches involved in Natural Language Processing (NLP) tasks have undergone profound changes over the past decade, converging to generalized language models that represent words with dense vectors [17]. The first Large Language Model (LLM) was successfully trained in 2018 by exposing a model to a vast quantity of text, representing an enormous amount of language knowledge.

The term *Generative AI Language Model – GLM* is used in this paper, rather than *Large Language Model – LLM*, to call attention to the fact that, recently, several language models have been made available that are, by comparison, smaller than LLMs published in the past and yet achieve excellent performance.

Different sets of attributes may characterize a GLM. The Open LLM Leaderboard[7] uses: *model type* – pretrained, continuously trained, fine-tuned on domain-specific datasets, chat models, etc.; *precision* – float16, bfloat16, 8bit, etc.; and *model size* in billions of parameters. Since models may not activate all parameters for a given request, the model size has to be qualified; for example, DBRX uses a fine-grained mixture-of-experts (MoE) architecture with a total of 132B parameters, of which 36B parameters are active on a given request [6].

The Independent Analysis of AI-Language Models and API Providers Web site[8] considers: *context length*, *model quality*, *price*, *throughput*, and *latency*. Context length refers to the number of tokens accepted as input/output, and limits the database metadata and data that can be passed to the GLM in the text-to-SQL task. Among the quality features, the Web site lists the *coding ability* of the model, which may be indicative of the text-to-SQL ability.

One may also list: *licensing mode*, where *proprietary* models run only on vendors' platforms, versus *open-source* models that can be downloaded and run

---

[7] https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
[8] https://artificialanalysis.ai

on a private platform, and the *release date* of the model, where more recent models are expected to have better performance.

Using just the model size, a classification of GLMs would be:

– *Small Language Model (SLM)*: the model typically has 10B parameters or less; an open-source small model can be downloaded and executed on a small server or a PC running a language model platform, such as LM Studio[9].
– *Medium Language Model (MLM)*: the model typically has more than 10B parameters, and at most 100B parameters; an open-source medium model can be downloaded and executed on a mid-range server, usually equipped with one or more GPUs; the server can be locally available or allocated as a cloud resource.
– *Large Language Model (LLM)*: the model typically has more than 100B parameters; a large model usually runs on a large-scale cloud-based infrastructure.

The licensing mode, model size, and context length stand out as relevant characteristics for the construction of natural language database interfaces based on text-to-SQL (see Table 1 for examples of recent GLMs). Indeed, the licensing mode determines where the model can be run, and a proprietary model running on the vendor's platform may raise data privacy concerns; the size directly influences the infrastructure requirements to run the model and achieve the desirable throughput and latency; and the context length has direct implications for the implementation of a text-to-SQL strategy since it limits the database metadata and data that can be passed to the GLM.

As mentioned in the introduction, the central question addressed in this paper is to assess how the model size affects the ability of a GLM to support the text-to-SQL task. This question is of tantamount importance for implementing GLM-based Natural Language database interfaces.

2.2 Text-to-SQL Benchmarks

The Spider – Yale Semantic Parsing and Text-to-SQL Challenge [25] defines 200 datasets, covering 138 domains, for training and testing text-to-SQL tools. For each database, Spider lists 20–50 hand-written NL questions and their SQL translations. An NL question Q_N, with an SQL translation Q_S, is classified as easy, medium, hard, and extra-hard, where the difficulty is based on the number of SQL constructs of Q_S – GROUP BY, ORDER BY, INTERSECT, nested sub-queries, column selections, and aggregators – so that an NL query whose translation Q contains more SQL constructs is considered harder. The set of NL questions introduced in Section 4 follows this classification.

Most databases in Spider have small schemas: the largest five databases have between 16 and 25 tables, and about half of the databases have schemas with five or fewer tables. Also, all Spider NL questions are phrased in terms used in the database schemas. These two features simplify the text-to-SQL task. Therefore,

---

[9] https://lmstudio.ai

Table 1. Examples of recent GLMs.

Owner	Model	License	Size	Context Length	Release Date	Reference
Meta	LLaMA-3-8B-instruct	Open	8B	8.18K	Apr 18, 2024	[3]
	LLaMA-3-70B-instruct	Open	70B	8.18K		
MS	Phi-3-mini	Open	3.8B	4K and 128K		
	Phi-3-small	Open	7B	4K and 128K	Apr 23, 2024	[1]
	Phi-3-medium	Open	14B	8K		
DataBricks	DBRX	Open	132B	32K	Mar 27, 2024	[6]
Google	Gemma 7B	Open	7B	8.19K	Feb 21, 2024	[12]
	Gemma 2	Open	27B	–	May 14, 2024	[24]
	Gemini 1.5 Pro	Proprietary	N/A	1.0M	Mar 08, 2024	[11]
Anthropic	Claude 3 Haiku	Proprietary	N/A	200K		
	Claude 3 Sonnet	Proprietary	N/A	200K	Mar 04, 2024	[4]
	Claude 3 Opus	Proprietary	N/A	200K		
OpenAI	GPT-4o	Proprietary	N/A	128K	May 13, 2024	
	GPT-4-turbo	Proprietary	N/A	128K	Apr 09, 2024	[21]
	GPT-3.5-turbo-0125	Proprietary	N/A	16K	Jan 25, 2024	

the results reported in the Spider leaderboard are biased toward databases with small schemas, and NL questions written in the schema vocabulary, which is not what one finds in real-world applications.

Spider has two interesting variations. Spider-Syn [8] is used to test how well text-to-SQL tools handle synonym, and Spider-DK [9] addresses testing how well text-to-SQL tools deal with domain knowledge.

BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation [16] is a large-scale cross-domain text-to-SQL benchmark in English. The dataset contains 12,751 text-to-SQL data pairs and 95 databases with a total size of 33.4 GB across 37 domains. BIRD addresses real-world applications by exploring three additional challenges: dealing with large and messy database values, external knowledge inference, and optimizing SQL execution efficiency. However, BIRD still does not have many databases with large schemas: of the 73 databases in the training dataset, only two have more than 25 tables, and, of the 11 databases used for development, the largest one has only 13 tables. Again, all NL questions are phrased based on terms used in the database schemas.

WikiSQL [26] has 80,654 NL sentences and SQL annotations of 24,241 tables. Each query in WikiSQL is limited to the same table and does not contain complex operations such as sorting and grouping.

Despite the availability of these benchmarks for text-to-SQL, and inspired by them, Section 4 describes a benchmark based on the Mondial database, a set of 100 test NL questions, with their gold-standard SQL translations, and a synthetic dataset (see Section 3). The reasons for introducing the Mondial benchmark are threefold. Mondial features a relational schema which is much larger than the schemas of most of the databases in Spider and BIRD, which poses significant challenges to text-to-SQL strategies. The Mondial database is populated with real data, which allows using an RAG technique that proved effective (see Section 3). Lastly, the test NL questions mimic those posed by real users and cover a wide range of SQL constructs.

2.3   Prompt Strategies for Text-to-SQL

The Spider Web site[10] publishes a leaderboard with the best-performing text-to-SQL tools. The top 5 tools listed in the published leaderboard are based on Prompt strategies and achieve an accuracy that range from an impressive 85.3% to 91.2% (two of the tools are not openly documented). Four tools use GPT-4, as their names imply. The three tools that provide detailed documentation have an elaborate first prompt that tries to select the tables and columns that best matches the NL question.

The BIRD Web site[11] also publishes a leaderboard with the best-performing tools. At the time of writing, the topmost two tools use GPT-4.

The Awesome Text2SQL Web site[12] lists the best-performing text-to-SQL tools on WikiSQL, Spider (Exact Match and Exact Execution) and BIRD (Valid Efficiency Score and Execution Accuracy). The "60 Top AI Text To SQL Bot Tools" Web site[13] lists other AI tools for text-to-SQL.

Finally, LangChain is a generic framework that offers several predefined Prompt strategies for text-to-SQL. Section 3 provides a few more details about the LangChain's SQLQueryChain, which proved to be cost-effective in [19], vis-a-vis the strategies at the top of the Spider leaderboard. Choosing a single text-to-SQL strategy, with the GLM as a flexibilization point, reduces the number of experiments, and permits concentrating on comparing the ability of different GLMs to generate SQL code.

2.4   Fine-tuning LLMs for text-to-SQL

Attempts to fine-tune open-source LLMs for the text-to-SQL task using NL question/SQL query pairs were reported in [10]. However, until recently, the fine-tuned models did not achieve a performance on Spider comparable to the zero-shot (i.e., with no examples) performance of GPT-3.5-turbo.

The Defog SQLCoder[14] is based on models fine-tuned for the text-to-SQL task. The latest version, sqlcoder-70b-alpha, features 70B parameters and was fine-tuned on a base StarCoder model on more than 20,000 human-curated questions, classified as in Spider, based on ten different schemas. The training dataset[15] consisted of prompt-completion pairs, encompassing several schemas with varying difficulty levels, whereas the evaluation dataset featured questions from novel schemas. The use of complex schemas, with 4-20 tables, challenged the model. The fine-tuning process occurred in two stages: the base model was first refined using easy and medium questions and then further fine-tuned on hard and extra-hard questions to yield the final model. The accuracy of the

---

[10] https://yale-lily.github.io/spider
[11] https://bird-bench.github.io
[12] https://github.com/eosphoros-ai/Awesome-Text2SQL
[13] https://topai.tools/s/Text-to-SQL-bot
[14] https://github.com/defog-ai/sqlcoder?tab=readme-ov-file
[15] https://defog.ai/blog/open-sourcing-sqlcoder/

sqlcoder-70b-alpha model on the Defog's dataset achieved 93.0%, whereas GPT-4 (Feb. 5, 2024) achieved 86%. Still, the experiments were based on databases with very small schemas.

DTS-SQL [23] is a two-stage fine-tuning process that separates schema linking and SQL generation. The authors first applied DTS-SQL to fine-tune Mistral-7B [14] and DeepSeek-7B [7] for text-to-SQL. Then, they evaluated the fine-tuned models on Spider and Spider-Syn and showed that the execution accuracy was improved by 3% to 7%. The fine-tuned DeepSeek-7B performed comparably to the best strategies in the Spider leaderboard. However, these results reflect the limitations of Spider, namely, that the databases have simple schemas.

Natural-SQL-7B by ChatDB[16] was fine-tuned for text-to-SQL from deepseek-coder-6.7b-instruct, using 8,000 pairs of NL questions / SQL queries (for PostgreSQL). It is part of the NaturalSQL by ChatDB[17] collection of models.

Using a proprietary, real-world database with a large schema, early experiments showed that Mistral-7B frequently hallucinated, and Code Llama 2-13B-instruct-text2sql[18] had poor performance, even using RAG (see Section 2.5). This may reflect the fact that Code Llama 2-13B-instruct-text2sql was trained on Spider and WikiSQL, again on databases with small schemas.

Finally, these last experiments, including the DTS-SQL fine-tuning, required only modest hardware, and showed that it is feasible to run SLMs on small local servers and yet achieve state-of-the-art performance on Spider. But, fine-tuned tools proved to be less than satisfactory for databases with larger schemas [19].

2.5 Retrieval-Augmented Generation (RAG) for Text-to-SQL

Retrieval-Augmented Generation (RAG), introduced in [15], is a strategy to incorporate data from external sources. This process ensures that the responses are grounded in retrieved evidence, thereby significantly enhancing the accuracy and relevance of the output.

In the context of text-to-SQL, recent RAG references include a technique for an LLM-based Text-to-SQL framework involving sample-aware prompting and a dynamic revision chain [13]. A RAG technique is used in [22] to retrieve the table and column descriptions from a metadata store to ensure that the NL question is related to the right tables and columns.

Section 3 outlines an approach that, given a relational database D_R, its schema D_S, and associated documentation D_{doc}, generates a synthetic dataset E of pairs (Q_N, Q_S), where Q_N is an NL question and Q_S is its SQL translation, using D_R, D_S and D_{doc}. The synthetic dataset E is then used in a RAG technique to improve the accuracy of text-to-SQL strategies on D_R. This approach is independent of the GLM adopted, that is, it can be combined with any GLM. The approach contrasts with the model fine-tuning strategy outlined in Section

---

[16] https://huggingface.co/chatdb/natural-sql-7b-GGUF
[17] https://github.com/cfahlgren1/natural-sql
[18] https://huggingface.co/support-pvelocity/Code-Llama-2-13B-instruct-text2sql-GGUF

2.4, which is model-specific but independent of the database. Also, the RAG technique may be combined with different text-to-SQL prompt strategies; the experiments in Section 5 combine SQLQueryChain with this RAG technique.

3   A Prototype Text-to-SQL Framework

This section outlines a prototype text-to-SQL framework based on a prompt strategy, SQLQueryChain[19], combined with a Retrieval-Augmented Generation (RAG) technique, leaving as flexibilization points the GLM and the database (the synthetic dataset is constructed from the database and is not considered as a flexibilization point here).

Figure 1 depicts the text-to-SQL framework, with the flexibilization points indicated in light grey. The framework has roughly the following steps:

Pre-processing

P1. (Sample Generation) Generate pairs (Q_i, S_i) from the database DB, where Q_i is an NL question and S_i is the corresponding SQL query.
P2. (Embedding) Embed each NL question Q_i into a vector V_i and store V_i along with the pair (Q_i, S_i) in a *synthetic dataset SD*.

Processing

S1. (Sample Retrieval) Given an NL question Q_0, embed Q_0 in a vector V_{Q_0}; retrieve from DS the top-k vectors V_i, $i \in [1, k]$, most similar to V_{Q_0}; create a context C with the pairs (Q_i, S_i), $i \in [1, k]$.
S2. (Schema Linking) Retrieve from DB a description of the tables and columns required to process Q_0; add these descriptions to C.
S3. (SQL Translation) Ask the GLM to translate Q_0 into an SQL query S_0 under the context C.
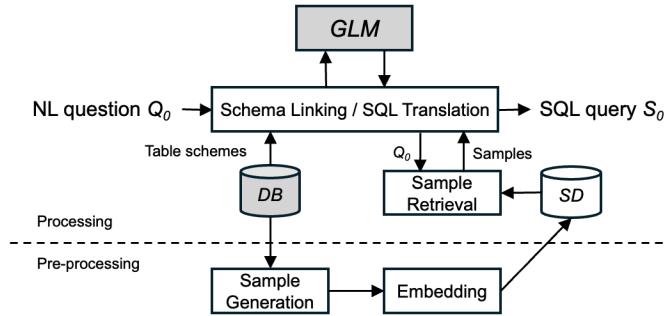

Fig. 1. The architecture of a prototype text-to-SQL framework.

---

[19] https://python.langchain.com/docs/use_cases/sql/prompting

SQLQueryChain, adopted in this paper, is the simplest LangChain text-to-SQL chain and is adapted in the framework to care for Steps S2 and S3. It trivializes the Schema Linking step by adding to the context C a description of all tables in the database schema. SQLQueryChain proved effective for real-world databases [5, 19], vis-a-vis much more complex text-to-SQL strategies listed at the top of the Spider and BIRD leaderboards.

The RAG technique adopted injects knowledge of the database schema and the data semantics into the SQL Translation step, and led to a considerable improvement of the accuracy of text-to-SQL Prompt strategies [5].

Steps P1 and P2 must be run only once for each database to generate the synthetic dataset SD in such a way as to improve the performance of the Schema Linking and SQL Translation steps. The technique introduced in [5] provides a robust strategy to construct database-specific synthetic datasets, especially when it comes to conveying the data semantics of the database to the GLM. The generation of SD is based on an algorithm that samples the database DB, its schema, and associated documentation, and calls a GLM to create an NL question Q_i from the sampled data and to translate Q_i into an SQL query S_i. Roughly, by varying how the sampling works, the pairs in SD provide SQL examples illustrating how the database schema is structured, how the user's language maps to the database schema, and how NL language constructions map to data values. Therefore, the synthetic dataset SD is specific to the database, but the algorithm is generic and applicable to any database. Table 3, discussed in Section 4.2, shows a sample of the synthetic dataset for Mondial.

This basic RAG technique has a limitation, though, since the Retrieval Step often returns pairs (Q_i, S_i) whose SQL queries are similar. The net effect is that the examples passed in the context C have little diversity. The *RAG technique with Question Decomposition* tries to remedy this limitation by decomposing the NL question Q_0 into sub-questions $Q_1, ..., Q_n$, which are then used in the Sample Retrieval step.

This paper then introduces a revised version of the Sample Retrieval step, not addressed in [5], and implemented using LangChain Question Decomposition:

S0. (Question Decomposition) Given an NL question Q_0, decompose Q_0 into sub-questions $Q_1, ..., Q_n$.
S1. (Sample Retrieval) Embed $Q_1, ..., Q_n$ into *base vectors* $V_{Q_1}, ..., V_{Q_n}$.
    Retrieve from DS the top-m vectors $V_{i,j}$ most similar to each base vector V_{Q_i}, $i \in [1, n]$, $j \in [1, m]$; order all retrieved vectors in decreasing order of similarity to their base vector and retain only the top-k vectors U_p, $p \in [1, k]$. For each $p \in [1, k]$, add to the context C the pair (Q_p, S_p) in DS associated with U_p.

4   The Mondial Benchmark

The Mondial benchmark[20] is based on the openly-available Mondial database, a set of 100 NL questions, and a synthetic dataset.

---

[20] Available at https://github.com/dudursn/text_to_sql_chatgpt_real_world/

4.1   The Mondial Database and the Set of NL Questions

The Mondial database stores geographic data and is an open-source suitable proxy for the proprietary industrial database that motivated this paper, but could not be made openly available. The relational version, adopted in this paper, has a total of 47,699 instances and features a schema with 46 tables, a total of 184 columns, and 49 foreign keys (some of which are multi-column)[21]. The Mondial schema is sophisticated but uses familiar terms, such as countries, cities, rivers, etc. A Mondial version with 34 tables is also part of the BIRD benchmark.

As introduced in [19], the benchmark contains a set of 100 NL questions and their gold-standard SQL queries, collected from the Internet or defined manually over the Mondial relational schema; the execution of a gold-standard SQL query returns the expected answer to the corresponding NL question. The NL questions are manually classified into *simple*, *medium*, and *complex*, that correspond to the easy, medium, and hard classes of the Spider benchmark. The difficulty of an NL question Q is based on the number of constructs of the corresponding gold-standard SQL query S, so that Q is considered harder, if S contains more constructs (GROUP BY, ORDER BY, INTERSECT, nested subqueries, column selections, and aggregators). The list of NL questions contains 33 simple, 33 medium, and 34 complex.

Table 2 shows an example of each type of NL question Q and its gold-standard SQL translation S. Note that, in the examples, S has just a restriction, when Q is simple, S has a join and a restriction, when Q is medium, and S has two joins and an aggregation, when Q is complex.

Table 2. A sample of the benchmark NL questions and their translations to SQL.

Type	NL Question	Gold-standard SQL Query	ID
simple	Show the Airports with elevation more than 3000.	`SELECT NAME FROM MONDIAL_AIRPORT` `WHERE ELEVATION > 3000.0`	33

medium	What type of government does Iran have?	`SELECT p.GOVERNMENT`
		`FROM MONDIAL_COUNTRY c`
		`INNER JOIN MONDIAL_POLITICS p`
		` ON p.COUNTRY = c.CODE`
		`WHERE c.NAME = "Iran"`
			99

complex	How much area do the countries that are adjacent to (or encompassed by) the Caribbean Sea cover in total?	`SELECT SUM(DISTINCT c.area)`
		` AS total_area`
		`FROM mondial_Country c`
		`JOIN mondial_encompasses e`
		` ON c.code = e.country`
		`JOIN mondial_geo_sea gs`
		` ON e.country = gs.country`
		`WHERE gs.sea = "Caribbean Sea"`
			24

4.2   The Mondial Synthetic Dataset

The Mondial benchmark also includes a synthetic dataset with 60,000 pairs, obtained using the technique outlined in Section 3 and adopting GPT-4. The

---

[21] https://relational.fit.cvut.cz/dataset/Mondial

construction of the synthetic dataset does not take into account the list of test NL questions. As such, the synthetic dataset should be useful to process any NL question the user submits to Mondial.

Table 3 shows examples of the synthetic dataset for Mondial. Table 3(a) corresponds to the sampled pair ("`MONDIAL_LAKE.AREA`", 500), that is, to the `AREA` column of table `MONDIAL_LAKE`, sampled from the database schema, and the value 500, sampled from the set of the `AREA` column values. The technique generated three different NL questions, which are paraphrases of each other, and mapped them to the same SQL query (repeated in the last column).

Table 3(b) shows a second example generated by sampling two columns: `ESTABLISHED` of `MONDIAL_ORGANIZATION` and `ELEVATION` of `MONDIAL_CITY`. Sampling two columns leads to a join between the two tables, reflected in the SQL query in the last column of the table.

Table 3. Examples of the synthetic dataset for Mondial.

(a) A simple example generated from sampling a single column.

#	NL question	SQL query
1	What are the lakes in the MONDIAL_LAKE table with an area greater than 500?	SELECT NAME, AREA FROM MONDIAL_LAKE WHERE AREA >500
2	Which lakes have an area larger than 500?	SELECT NAME, AREA FROM MONDIAL_LAKE WHERE AREA >500
3	lakes area larger than 500	SELECT NAME, AREA FROM MONDIAL_LAKE WHERE AREA >500

(b) A more complex example generated from sampling two columns.

#	NL question	SQL query
1	What is the time range of ESTABLISHED for organizations in the MONDIAL_ORGANIZATION table where the corresponding cities in the MONDIAL_CITY table have a non-null ELEVATION value?	SELECT MIN(MO.ESTABLISHED), MAX(MO.ESTABLISHED) FROM MONDIAL_ORGANIZATION MO JOIN MONDIAL_CITY MC ON MO.CITY = MC.NAME WHERE MC.ELEVATION IS NOT NULL
2	What is the time period when organizations were established in cities with known elevation values?	SELECT MIN(MO.ESTABLISHED), MAX(MO.ESTABLISHED) FROM MONDIAL_ORGANIZATION MO JOIN MONDIAL_CITY MC ON MO.CITY = MC.NAME WHERE MC.ELEVATION IS NOT NULL
3	time period organizations established cities known elevation values	SELECT MIN(MO.ESTABLISHED), MAX(MO.ESTABLISHED) FROM MONDIAL_ORGANIZATION MO JOIN MONDIAL_CITY MC ON MO.CITY = MC.NAME WHERE MC.ELEVATION IS NOT NULL

The rest of this section uses the Mondial benchmark to briefly illustrate the effect of Question Decomposition on Sample Retrieval.

Consider the following NL question in the Mondial benchmark:

Q_{17} : "Select cities whose population is greater than 100000, altitude greater than 2500, and the countries they belong to have population growth greater than 1."

Question Decomposition produces the following sub-questions for Q_{17}:

$Q_{17.A}$: "What are the cities with a population greater than 100000?"
$Q_{17.B}$: "What are the cities with an altitude greater than 2500?"
$Q_{17.C}$: "What are the countries with a population growth greater than 1?"

Table 4 shows some examples of NL Question/SQL Query pairs retrieved from the synthetic dataset for these sub-questions.

Table 4. Some examples retrieved for the sub-questions of Q_{17}.

Sub-Question	Examples Retrieved	
	NL Question	**SQL Query**
$Q_{17.A}$	countries cities population more than 100,000	SELECT DISTINCT MC.NAME FROM MONDIAL_COUNTRY MC JOIN MONDIAL_CITY MCI ON MC.CODE = MCI.COUNTRY WHERE MCI.POPULATION > 100000
$Q_{17.B}$	countries GDP higher than 100,000 and cities elevation greater than 1000 meters	SELECT E.COUNTRY, E.GDP, C.NAME, C.ELEVATION FROM MONDIAL_ECONOMY E JOIN MONDIAL_CITY C ON E.COUNTRY = C.COUNTRY WHERE E.GDP > 100000 AND C.ELEVATION > 1000
$Q_{17.C}$	Which countries have a status of 'member' and a population growth rate higher than 0.5?	SELECT MC.NAME FROM MONDIAL_ISMEMBER MI JOIN MONDIAL_COUNTRY MC ON MI.COUNTRY = MC.CODE JOIN MONDIAL_POPULATION MP ON MI.COUNTRY = MP.COUNTRY WHERE MI.TYPE = 'member' ANDMP.POPULATION_GROWTH > 0.5

Table 5 presents the gold-standard SQL query corresponding to Q_{17}, and the SQL queries generated with and without Question Decomposition using GPT-4. Without Question Decomposition, GPT-4 joined just tables `CITY` and `COUNTRY`, and hallucinated, creating a restriction on column `population_growth` of table `COUNTRY`, which does not exist. With Question Decomposition, the NL question was broken into three simpler sub-questions, one for each restriction, and Sample Retrieval returned examples that helped GPT-4 correctly synthesize the required joins and restrictions.

Table 5. Comparison of SQL queries for Q_{17}.

NL Question	Gold-standard SQL Query	SQL Query generated without decomposition	SQL Query generated with decomposition
Select cities whose population is greater than 100000, altitude greater than 2500, and the countries they belong to have population growth greater than 1.	SELECT ci.name FROM mondial_city ci INNER JOIN mondial_country c ON c.code = ci.country INNER JOIN mondial_population p ON p.country = c.code WHERE ci.population > 100000 AND ci.elevation > 2500 AND p.population_growth > 1	SELECT city.name, city.population, city.elevation, country.population_growth FROM city JOIN country ON city.country = country.code WHERE city.population > 100000 AND city.elevation > 2500 AND country.population_growth > 1	SELECT c.name, c.population, c.elevation, p.population_growth FROM MONDIAL_CITY c JOIN MONDIAL_COUNTRY co ON c.country = co.code JOIN MONDIAL_POPULATION p ON co.code = p.country WHERE c.population > 100000 AND c.elevation > 2500 AND p.population_growth > 1

5   Experiments

Setup configurations. All experiments used the text-to-SQL framework, described in Section 3, instantiated with the Mondial database. The experiments ran three versions of the framework, identified as:

- *SQLQueryChain*: just Steps S2 and S3 of the framework, that is, the standard SQLQueryChain. This is the baseline strategy.
- *SQLQueryChain with RAG*: Steps S1, S2, and S3, using the synthetic dataset with 60,000 pairs pre-computed for Mondial, defined in Section 4. The usual cosine similarity function was adopted to compare the user's NL question and the NL questions in the dataset. These experiments assess whether RAG improves accuracy for a given model.
- *SQLQueryChain with RAG and Question Decomposition*: Step S0, the modified Step S1, and Steps S2 and S3. These experiments assess whether Question Decomposition further improves accuracy for a given model, as proposed in this paper.

The final SQL queries were then run on the Mondial database. In all three versions, the GLM remains the only open flexibilization point of the framework since the experiments aimed to test different GLMs under the same conditions.

The experiments then instantiated the framework with the following models, running on different platforms:

- *Proprietary Large Language Models:* GPT-3.5-turbo-16K and GPT-4, running on the OpenAI (paid) platform.
- *Open-source Medium Language Models:* LLaMA-3-70B-instruct and DBRX, running on a server configured on the Azure platform.
- *Open-source Small Language Models:* Gemma 7B and Natural-SQL-7B by ChatDB, running locally on LM Studio.

Natural-SQL-7B by ChatDB was selected because it was fine-tuned for text-to-SQL and had good results. Gemma 7B was just released at the time of writing. The other models were chosen because they performed well on a coding task.

Performance Indicator. The experiments tested the *accuracy*, defined as the number of correct predicted SQL queries divided by the total number of SQL queries, as usual.

The experiments used an automated procedure to compare the *predicted* and the *gold-standard* SQL queries, entirely based on column and table values, and not just column and table names. The results of the automated procedure were manually checked to eliminate false negatives, caused mainly by mismatches in the target clause when there were multiple alternatives to identify entities.

Results. Recall that the Mondial benchmark has 100 NL questions, where 33 are simple, 33 are medium, and 34 are complex.

Table 6 summarizes the results for the selected models. Note that the lines are grouped according to the alternative of the framework used. Also, note that the experiments ran the SQLQueryChain with RAG and Question Decomposition alternative only for each of the small, medium, and large models that had the best accuracy in the SQLQueryChain with the RAG alternative. The results in Table 6 are contributions of this paper, except for Lines 1 and 2, published in [19], and Lines 7 and 8, published in [5].

Table 6. Results for the different models – Mondial Benchmark.

#	Size	Strategy / Model	#Correct Predicted Queries				Accuracy			
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Total
		SQLQueryChain								
1	L	GPT-3.5-turbo-16k	25	22	13	60	0,76	0,67	0,38	0,60
2	L	GPT-4	27	28	15	70	0,82	0,85	0,44	0,70
3	M	LLaMA-3-70B-instruct	22	21	23	66	0,67	0,64	0,68	0,66
4	M	DBRX	4	2	6	11	0,12	0,06	0,15	0,11
5	S	Natural-SQL-7B by ChatDB	28	25	11	64	0,85	0,76	0,32	0,64
6	S	Gemma 7B	12	7	2	21	0,36	0,21	0,06	0,21
		SQLQueryChain with RAG								
7	L	GPT-3.5-turbo-16K	28	23	21	72	0,85	0,70	0,62	0,72
8	L	GPT-4	28	31	27	86	0,85	0,94	**0,79**	0,86
9	M	LLaMA-3-70B-instruct	29	29	22	80	0,88	0,88	0,65	0,80
10	M	DBRX	21	15	10	46	0,64	0,45	0,29	0,46
11	S	Natural-SQL-7B by ChatDB	29	18	11	58	0,88	0,55	0,32	0,58
12	S	Gemma 7B	14	6	5	25	0,42	0,18	0,15	0,25
		SQLQueryChain with RAG and Question Decomposition								
13	L	GPT-4	32	32	25	89	**0,97**	**0,97**	0,74	**0,89**
14	M	LLaMA-3-70B-instruct	28	27	25	80	0,85	0,81	0,73	0,80
15	S	Natural-SQL-7B by ChatDB	27	19	16	62	0,82	0,58	0,47	0,62

Results for the Large Language Models. Lines 1 and 7 show the results for the framework instantiated with GPT-3.5-turbo-16K without RAG (Line 1) and with RAG (Line 7). The configuration with RAG achieved a total accuracy of 72%, surpassing all previous strategies tested in [18], except for the RAG-based technique with GPT-4 and C3 with GPT-4.

Lines 2, 8, and 13 show that, in all three alternatives, the framework instantiated with GPT-4 obtained the best total accuracy. The Query Decomposition technique led to a total accuracy (Line 13) comparable to the state-of-the-art tools reported in the Spider and BIRD leaderboards, recalling again that the Mondial benchmark is far more challenging than any of the databases used in Spider or BIRD. In particular, this configuration correctly translated 97% of the simple and medium NL questions, which is a remarkable achievement. However, GPT-4 is proprietary and runs on the OpenAI (paid) platform.

Results for the Medium Language Models. Lines 3, 9, and 14 correspond to the results of the framework instantiated with LLaMA-3-70B-instruct. Comparing LaMA-3-70B-instruct (Line 9) and GPT-3.5-turbo-16K (Line 7), the former achieved a better total accuracy (80%) and better accuracies in all types of NL questions. When comparing the RAG-only (Line 9) and the RAG with Query Decomposition techniques (Line 14), the total accuracy remained the same, but Query Decomposition improved the accuracy for the complex NL questions.

By contrast, Lines 4 and 10 indicate that DBRX had poor accuracy, which suggests that this model is not suited for text-to-SQL, although it performed well on a coding task.

These results suggest that an open-source, medium-sized model may replace a proprietary large-sized model. They imply that an NL database interface may run, with comparable accuracy, on a private platform, rather than on a proprietary platform, which may mitigate data privacy concerns.

Results for the Small Language Models. Lines 5, 11, and 15 show the results for the framework instantiated with Natural-SQL-7B by ChatDB. The model achieved the second-best accuracy (88%) among all models and alternatives for simple NL queries, surpassed only by GPT-4 with RAG with Query Decomposition. However, note that the use of the RAG technique substantially decreased the accuracy for medium NL questions and did not increase the accuracy for complex NL queries. By inspecting the sample SQL queries retrieved, one observes that the examples were of poor quality and referred to tables not directly related to input the NL question, which presumably confused the model and led it to generate incorrect SQL queries.

Lines 6 and 12 show the results for the text-to-SQL framework instantiated with Gemma 7B without RAG (Line 6) and with RAG (Line 12). The performance of the model was significantly less than that of Natural-SQL-7B by ChatDB. The Phi-3-mini-128K, Mistral-7B-Instruct-text2sql, and Code Llama 2-13B-instruct-text2-sql models were also tested, but had very poor performance, which indicates that these models are not a suitable choice.

These results suggest that a Small Language Model fine-tuned for text-to-SQL, running on a local small server, is a competitive option when the NL questions are mostly simple.

Additional comments. The RAG technique of Section 3 injects knowledge of the database schema and the data semantics into a GLM. Overall, the results suggest that accuracy is improved for large and medium models but not necessarily for small ones. This is an effect that has to be further investigated. Apparently, the Sample Retrieval step fetched pairs whose SQL queries had tables unrelated to the user NL question, which confused the GLM.

In general, an analysis of the incorrect predicted SQL queries uncovered that the GLMs had difficulties interpreting the Mondial schema, which indicates that the database designer should revise the schema to facilitate the Schema Linking step given the expected NL questions. The designer should also consider creating views that introduce some redundancies to reduce the number of joins required to translate the user NL questions. For example, quite a few of the incorrect predicted SQL queries reflect a simple problem – several Mondial tables, such as

```
MONDIAL_CITY(NAME, COUNTRY, PROVINCE, POPULATION, LATITUDE, LONGITUDE, ELEVATION)
```

have a column named `COUNTRY` that is populated with the Country code, not the Country name. By contrast, the user NL questions refer to Countries by their names, not by their codes. This can be easily fixed by creating a view that includes a predefined column, `COUNTRY_NAME`, such as

```
MONDIAL_CITY_VIEW(NAME, COUNTRY_NAME, COUNTRY_CODE, PROVINCE, POPULATION, ...)
```

This simple solution proved quite effective for a real-world private database with a large schema that was not considered *LLM-friendly* [20].

6   Conclusions and Directions for Future Work

This paper investigated how the model size affects the ability of a Generative AI Language Model (GLM) to support the text-to-SQL task for databases with sophisticated schemas typical of real-world applications. The paper described experiments using a text-to-SQL framework, instantiated with the Mondial database and a synthetic dataset with 60,000 pairs, pre-computed for Mondial. The experiments tested GLMs of different sizes under the same conditions.

The results suggest that, for a database with a sophisticated schema and a set of 100 challenging NL questions:

1. The text-to-SQL framework obtained the best results with GPT-4, as expected from previous results [5, 19].
2. When compared with SQLQueryChain, the baseline, the RAG technique with Question Decomposition led to a significant improvement of the total accuracy for large and medium models, but not necessarily for small models.
3. An open-source medium-sized model may achieve an accuracy similar to a proprietary large-sized model.
4. An open-source small-sized model fine-tuned for text-to-SQL, running on a small local server, is a competitive option when the NL questions are simple.

The results reported in the paper, therefore, suggest that an open-source medium-sized model, coupled with a RAG technique with Question Decomposition, can achieve sufficient performance to provide the basis for an NL interface for a real-world database that may run on a private platform rather than on a proprietary platform, which may mitigate data privacy concerns.

When dealing with a Natural Language database interface based on a language model, a database designer should also worry about the design of a synthetic dataset for the database in question, as outlined in Sections 3 and 4.2, to pass knowledge of the database schema and the data semantics to the language model. That is, the database designer should be concerned with exposing the database metadata and the data semantics to the language model, and not just to the programmers or end-users, which is a task different from traditional database design. However, when the problem is a mismatch between the vocabulary of the user NL questions and the vocabulary induced by the database schema, simply renaming the tables and columns or introducing views would be a viable solution, as illustrated at the end of Section 5.

There are multiple paths for future work. New, more powerful language models emerge with a high frequency. These models should be considered for the text-to-SQL task on real-world databases, using the Mondial benchmark and the framework introduced in this paper as a testbed. In another direction, the technique to create a synthetic dataset for a given database can be enhanced to generate more accurate samples. Lastly, the ambiguity problem, inherent to NL questions, must be addressed, perhaps using a disambiguation dialog.

References

1. Abdin, M.I., et al.: Phi-3 technical report: A highly capable language model locally on your phone. Tech. Rep. MSR-TR-2024-12, Microsoft (April 2024), https://www.microsoft.com/en-us/research/publication/phi-3-technical-report-a-highly-capable-language-model-locally-on-your-phone/

2. Affolter, K., Stockinger, K., Bernstein, A.: A comparative survey of recent natural language interfaces for databases. The VLDB Journal **28** (Aug 2019), https://doi.org/10.1007/s00778-019-00567-8

3. AI@Meta: Llama 3 model card. Tech. rep., Meta (2024), https://github.com/meta-llama/llama3/blob/main/MODEL_CARD.md

4. Anthropic: Introducing the next generation of claude (2024), https://www.anthropic.com/news/claude-3-family

5. Coelho, G., Nascimento, E.R., Izquierdo, Y.T., Garcia, G.M., Feijó, L., Lemos, M., Garcia, R.L., R. de Oliveira, A., Pinheiro, J., Casanova, M.A.: Improving the accuracy of text-to-sql tools based on large language models for real-world relational databases. In: C. Strauss et al. (Eds.): Proceedings of the 35th International Conference on Database and Expert Systems Applications, DEXA 2024, LNCS 14910. pp. 1–25. Springer (2024). https://doi.org/10.1007/978-3-031-68309-1_8

6. DataBricks: Introducing dbrx: A new state-of-the-art open llm (2024), https://platform.openai.com/docs/models/overview

7. DeepSeek-AI, et al.: Deepseek llm: Scaling open-source language models with longtermism (2024), https://doi.org/10.48550/arXiv.2401.02954

8. Gan, Y., Chen, X., Huang, Q., Purver, M., Woodward, J.R., Xie, J., Huang, P.: Towards robustness of text-to-sql models against synonym substitution. CoRR **abs/2106.01065** (2021), https://doi.org/10.48550/arXiv.2106.01065

9. Gan, Y., Chen, X., Purver, M.: Exploring underexplored limitations of cross-domain text-to-sql generalization. In: Proc. 2021 Conf. on Empirical Methods in Natural Language Processing. pp. 8926–8931 (2021), https://doi.org/10.18653/v1/2021.emnlp-main.702

10. Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-sql empowered by large language models: A benchmark evaluation. Proc. VLDB Endow. **17**(5), 1132–1145 (may 2024). https://doi.org/10.14778/3641204.3641221

11. GeminiTeam: Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context (2024), https://storage.googleapis.com/deepmind-media/gemini/gemini_v1_5_report.pdf

12. GemmaTeam: Gemma: Open models based on gemini research and technology. arXiv preprint (2024), https://doi.org/10.48550/arXiv.2403.08295

13. Guo, C., Tian, Z., Tang, J., Li, S., Wen, Z., Wang, K., Wang, T.: Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. In: Luo, B., Cheng, L., Wu, Z.G., Li, H., Li, C. (eds.) Neural Information Processing. pp. 341–356. Springer Nature Singapore, Singapore (2024)

14. Jiang, A.Q., et al.: Mistral 7b. arXiv preprint (2023), https://doi.org/10.48550/arXiv.2310.06825

15. Lewis, P., et al.: Retrieval-augmented generation for knowledge-intensive nlp tasks. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 9459–9474. Curran Associates, Inc. (2020), https://api.semanticscholar.org/CorpusID:218869575

16. Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K.C., Huang, F., Cheng, R., Li, Y.: Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. NIPS '23, Curran Associates Inc., Red Hook, NY, USA (2024)

17. Manning, C.D.: Human Language Understanding & Reasoning. Daedalus **151**(2), 127–138 (05 2022), https://doi.org/10.1162/daed_a_01905

18. Nascimento, E.R.: Querying Databases with Natural Language: The use of Large Language Models for Text-to-SQL tasks. Master's thesis, Dissertation presented to the Graduate Program in Informatics, PUC-Rio (2024)

19. Nascimento, E.R., Garcia, G.M., Feijó, L., Victorio, W.Z., Lemos, M., Izquierdo, Y.T., Garcia, R.L., Leme, L.A.P., Casanova, M.A.: Text-to-sql meets the real-world. In: Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1. pp. 61–72 (2024)

20. Nascimento, E.R., Izquierdo, Y.T., Garcia, G.M., Coelho, G., Feijó, L., Lemos, M., Leme, L.A.P., Casanova, M.A.: My database user is a large language model. In: Proceedings of the 26th International Conference on Enterprise Information Systems – Volume 1. pp. 800–806 (2024)

21. OpenAI: Openai models (2024), https://platform.openai.com/docs/models

22. Panda, S., Gozluklu, B.: Build a robust text-to-sql solution generating complex queries, self-correcting, and querying diverse data sources. AWS Machine Learning Blog (28 Feb 2024)

23. Pourreza, M., Rafiei, D.: Dts-sql: Decomposed text-to-sql with small large language models. arXiv preprint (2024), https://doi.org/10.48550/arXiv.2402.01117

24. Warkentin, T., Zhai, X., Peran, L.: Introducing paligemma, gemma 2, and an upgraded responsible ai toolkit. Tech. rep., Google (2024), https://developers.googleblog.com/en/gemma-family-and-toolkit-expansion-io-2024/

25. Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., Radev, D.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: Proc. 2018 Conf. on Empirical Methods in Natural Language Processing. pp. 3911–3921 (Oct – Nov 2018), https://doi.org/10.18653/v1/D18-1425

26. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. CoRR **abs/1709.00103** (2017), https://doi.org/10.48550/arXiv.1709.00103