Pontifícia Universidade Católica
do Rio de Janeiro

Aiko Ramalho Oliveira

Text-to-SQL on Real World Datasets

Trabalho de Conclusão de Curso

Final Project Report, presented to the Undergraduate of PUC-Rio as a partial requirement for obtaining the degree of Bachelor in Ciência da Computação

Advisor: Prof. Marco Antonio Casanova

Rio de Janeiro
July 2024

Pontifícia Universidade Católica
do Rio de Janeiro

Aiko Ramalho Oliveira

Text-to-SQL on Real World Datasets

Final Project Report, presented to the Undergraduate of PUC-Rio as a partial requirement for obtaining the degree of Bachelor in Ciência da Computação Approved by the Examination Committee:

Prof. Marco Antonio Casanova
Advisor
Departamento de Informática – PUC-Rio

Rio de Janeiro, July 19th, 2024

To my parents, for their support
and encouragement.

Acknowledgments

Firstly, I would like to express my deepest gratitude to my father, Sérgio N. Oliveira, to whose memory I dedicate this work. He instilled in me a love for PUC, mathematics, hard work, and the value of truly important things. The primary motivation for embarking on this work was his illness over the past three years and my strong desire for him to witness and take pride in this achievement. Due to higher reasons, the Supreme Intelligence called him back to the spiritual realm in March of this year, making this wish unfulfillable. I am immensely grateful for the investment he made in me over the 24 years we shared together physically, and I present this work with the conviction that he is watching, proud, from the spiritual plane.

I would also like to thank my mother and my sisters for shaping my entire educational journey, enabling me to reach this point. Especially to my mother, for her unconditional support throughout my life, regardless of the endeavors I chose to pursue. I extend my gratitude to the Sattamini de Souza family for welcoming me as one of their own. Lastly, I wish to thank my wife-to-be, Isadora S. de Souza, for her immeasurable support throughout the development of this work.

To PUC-Rio, TecGraf, and my advisor:
Thank you for all the academic support that made this work possible. Without you, it would never have been achievable.

Abstract

Oliveira, Aiko R.; Casanova, Marco (Advisor). **Text-to-SQL on Real World Datasets**. Rio de Janeiro, 2024. 44p. Trabalho de Conclusão de Curso – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

In the rapidly evolving field of Natural Language Processing (NLP), the task of translating natural language queries into SQL queries (Text-to-SQL) has garnered significant attention due to its potential to simplify database interactions for non-technical users. This final project, titled "Text-to-SQL on Real World Datasets," explores innovative methods to enhance the accuracy and efficiency of Text-to-SQL systems, specifically focusing on real-world databases with complex schemas.

The project leverages the Retrieval-Augmented Generation (RAG) technique to improve Text-to-SQL accuracy by integrating external data sources and fine-tuning strategies. A combination of synthetic dataset generation and prompt strategies is employed to enhance the model's performance. The Mondial dataset, known for its complexity and richness in geographic data, serves as the benchmark for evaluating the proposed techniques.

The study aims to develop a robust Text-to-SQL framework capable of handling diverse and complex queries, thereby making database interactions more intuitive and accessible. The methodologies, experiments, and findings documented in this report contribute valuable insights to ongoing research in NLP and database management systems.

Keywords

LLM; NLP; RAG; GLM; Text-To-SQL.

Resumo

Oliveira, Aiko R.; Casanova, Marco. **Text-to-SQL on Real World Datasets**. Rio de Janeiro, 2024. 44p. Trabalho de Conclusão de Curso – Departamento de Informática, Pontifícia Universidade Católica do Rio de Janeiro.

No campo em rápida evolução do Processamento de Linguagem Natural (NLP), a tarefa de traduzir consultas em linguagem natural para consultas SQL (Text-to-SQL) tem ganhado atenção significativa devido ao seu potencial para simplificar interações com bancos de dados para usuários não técnicos. Este projeto final, intitulado "Text-to-SQL em Conjuntos de Dados do Mundo Real," explora métodos inovadores para melhorar a precisão e a eficiência dos sistemas Text-to-SQL, focando especificamente em bancos de dados do mundo real com esquemas complexos.

O projeto utiliza a técnica de Geração Aumentada por Recuperação (RAG) para melhorar a precisão do Text-to-SQL, integrando fontes de dados externas e estratégias de ajuste fino. Uma combinação de geração de conjuntos de dados sintéticos e estratégias de prompts é empregada para aprimorar o desempenho do modelo. O conjunto de dados Mondial, conhecido por sua complexidade e riqueza em dados geográficos, serve como referência para avaliar as técnicas propostas.

O objetivo do estudo é desenvolver uma estrutura robusta de Text-to-SQL capaz de lidar com consultas diversas e complexas, tornando as interações com bancos de dados mais intuitivas e acessíveis. As metodologias, experimentos e descobertas documentadas neste relatório contribuem com insights valiosos para a pesquisa contínua em NLP e sistemas de gerenciamento de bancos de dados.

Palavras-chave

LLM; NLP; RAG; GLM; Text-To-SQL.

Table of contents

List of figures

List of tables

List of algorithms

List of codes

List of Abreviations

LLM - *Large Language Model*

NLP - *Natural Language Processing*

GLM - *Generative AI Language Model*

GPT - *Generative Pre-trained Transformer.*

RAG - *Retrieval-Augmented Generation*

NL - *Natural Language*

DB - *Database*

It's not about creating artificial intelligence,
but amplifying human intelligence.

Amber Case, *TED Talk.*

1 Introduction

In the rapidly evolving field of Natural Language Processing (NLP), the task of translating natural language queries into SQL queries (Text-to-SQL) has garnered significant attention due to its potential to simplify database interactions for non-technical users. This final project, titled "Text-to-SQL on Real World Datasets", explores innovative methods to enhance the accuracy and efficiency of Text-to-SQL systems, specifically focusing on real-world databases with complex schemas.

The journey of NLP approaches has transitioned through several eras, from simple rule-based models in the 1950s to sophisticated deep learning-based models in the present day. The advent of Large Language Models (LLMs) like GPT has revolutionized the field, enabling a more generalized understanding of language through self-supervised learning. Despite their prowess, LLMs face challenges such as temporal generalization and factual grounding, which are critical when dealing with dynamic and specific database queries.

This work leverages the Retrieval-Augmented Generation (RAG) technique to improve Text-to-SQL accuracy by integrating external data sources and fine-tuning strategies. The study employs a combination of synthetic dataset generation and prompt strategies to enhance the model's performance. Specifically, it utilizes the Mondial dataset, known for its complexity and richness in geographic data, to benchmark and evaluate the proposed techniques.

Through this project, the goal is to develop a robust Text-to-SQL framework that can handle diverse and complex queries, providing a significant step forward in making database interactions more intuitive and accessible. This report documents the methodologies, experiments, and findings, contributing valuable insights to the ongoing research in NLP and database management systems.

This document is structured as follows. In Chapter 2 we present some previous work relevant to our problem. In Chapter 3 we explain our proposal. In Chapter 4 we show our results. Finally, in Chapter 5 we present our conclusion and future work.

2 Background

2.1 The Emergence of Large Language Models

The approaches involved in Natural Language Processing (NLP) tasks have undergone deep changes over the past decades. (MANNING, 2022) classifies NLP approaches in roughly four eras. The first era, from 1950 to 1969, was characterized by simple rule-based models, achieving very limited results due to the lack of computation, data, and knowledge of human language structure. The second era, from 1970 to 1992, benefited from the rapid development of linguistic theories, allowing the creation of a new generation of hand-built systems from knowledge-based artificial intelligence models. The third era, from 1993 to 2012, saw a shift from knowledge-based to machine-learning models, when digital text became widely available, providing enough data to allow some level of language understanding by using mainly supervised machine-learning methods. The last era, from 2013 to the present, extended the use of machine learning, allowing a more generalized language understanding by using self-supervised learning and deep learning-based models to represent words with dense vectors. Within this context, the first LLM was successfully trained in 2018, by exposing a model to an extremely large quantity of text, allowing the representation of an enormous amount of language knowledge.

LLMs follow the transformer neural network architecture (VASWANI et al., 2017; TUNSTALL; WERRA; WOLF, 2022; JURAFSKY; MARTIN, 2023). The main intuition behind transformers is the use of the attention mechanism, where the representation of a given position is computed based on a weighted combination from others. Based on this architecture, the self-supervised strategy for LLMs usually involves masking words in the text, and training the model to predict the missing words from the outer context. After this process is extensively repeated, the model learns generalized syntactic structures of sentences.

As a consequence of their generalization capacity, LLMs perform well in a wide variety of tasks, to the point of shifting the focus of NLP research away from the previous paradigm of training specialized supervised models for the specific tasks. In fact, an LLM can be redeployed to a particular NLP task with a small number of further instructions.

However, an LLM reflects the data it was trained with. In particular, an LLM suffers from the "temporal generalization problem" – capturing facts that change over time – and the "factual grounding problem" – capturing specific

facts. To circumvent these limitations, the user may *fine-tune* the LLM, that is, retrain it with more examples, or he may adopt *few-shot learning*, that is, add a few examples in a dialog interaction so that the model can capture what the user is trying to do and generate a plausible completion. As mentioned in the introduction,in the context of databases, these challenges translate to: how to fine-tune an LLM to a specific database; and how to generate dialog context, defined with data stored in a database, for an LLM.

2.2 Generative AI Language Models

The term *Generative AI Language Models – GLM* is also used in this work, 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[1] 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 (DATABRICKS, 2024).

The Independent Analysis of AI-Language Models and API Providers Web site[2] 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 (see Table 2.1), which may be indicative of the text-to-SQL ability.

Table 2.1: Coding – HumanEval.

Model	Score
GPT-4-turbo	85.4
LLaMA-3-70B-instruct	81.7
GPT-3.5-turbo	73.2
Gemini 1.5 Pro	71.9
DBRX	70.1
Gemini 1.0 Pro	63.4
LLaMA-3-8B-instruct	62.2

Source: Independent analysis of AI language models and API providers.

[1]https://huggingface.co/spaces/HuggingFaceH4/open_llm_leaderboard
[2]https://artificialanalysis.ai

One may also list: *licensing mode*, where *proprietary* models run only on vendors' platforms, versus *open-source* models that can be downloaded and run 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 even a personal computer running a language model platform, such as LM Studio[3].
- *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, typically 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 is typically proprietary and 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 2.2 for examples of recent GLMs).

Table 2.2: Examples of recent GLMs.

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

[3]https://lmstudio.ai

2.3 Text-to-SQL Datasets

The Spider – Yale Semantic Parsing and Text-to-SQL Challenge (YU et al., 2018) offers datasets for training and testing text-to-SQL tools. Spider features nearly 200 databases covering 138 different domains from three resources: 70 complex databases from different college database courses, SQL tutorial Web sites, online CSV files, and textbook examples; 40 databases from DatabaseAnswers[4]; and 90 databases based on WikiSQL, with about 500 tables in about 90 different domains. For each database, Spider lists 20-50 hand-written NL questions and their SQL translations; the SQL queries cover all the major SQL components.

Spider proposes three evaluation metrics: *component matching* checks whether the components in the prediction and the ground truth match exactly; *exact matching* measures whether the predicted query as a whole is equivalent to the gold query; *execution accuracy* requires that the predicted SQL query select a list of gold values and fill them into the right slots.

One may criticize Spider for having many databases with very small schemas. The largest 5, in number of tables, are: `baseball_1`, with 25 tables, `cre_Drama_ Workshop_Groups`, with 18 tables, and `cre_Theme_park`, `imdb`, and `sakila_1`, with 16 tables. In fact, about half of the databases have schemas with five tables or less. Therefore, the results reported in the leaderboard are highly biased towards databases with small schemas and do not reflect real-world databases.

Spider has two interesting variations. Spider-Syn (GAN et al., 2021) is used to test how well text-to-SQL tools handle synonym, and Spider-DK (GAN; CHEN; PURVER, 2021) addresses testing how well text-to-SQL tools deal with domain knowledge.

BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation (LI et al., 2023) 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.

[4] http://www.databaseanswers.org/

WikiSQL (ZHONG; XIONG; SOCHER, 2017) 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.

SQL-Eval (PING, 2023) is a framework[5] that evaluates the correctness of text-to-SQL strategies, created during the development of Defog's SQLCoder.

Finally, the `sql-create-context`[6] dataset was built for the text-to-SQL task. It contains 78,577 examples of NL queries, SQL CREATE TABLE statements, and SQL Queries answering the questions. The CREATE TABLE statement provides context for the LLMs, without having to provide actual rows of data.

Despite the availability of these benchmarks for text-to-SQL, and inspired by them, Section 4.1.1 describes a benchmark tuned to the problem addressed in this study. The benchmark consists of a relational database, whose design is based on a real-world database, three sets of LLM-friendly views, specified as proposed in (NASCIMENTO et al., 2024b), and a set of 100 test NL questions, that mimic those posed by real users, and their ground truth SQL translations.

2.4 Text-to-SQL Prompt Strategies

The Spider Web site[7] 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.

Were (in descending order): "MiniSeek" (no reference available at the time of writing); "DAIL-SQL + GPT-4 + Self-Consistency" and "DAIL-SQL + GPT-4", both reported in (GAO et al., 2023); "DPG-SQL + GPT-4 + Self-Correction" (no reference available at the time of writing); "DIN-SQL + GPT-4" (POURREZA; RAFIEI, 2023); "Hindsight Chain of Thought with GPT-4" (no reference available at the time of writing); and "C3 + ChatGPT + Zero-Shot" (DONG et al., 2023).

The BIRD Web site[8] also publishes a leaderboard with the best-performing tools. At the time of writing, the topmost two tools use GPT-4.

[5] Available at https://github.com/defog-ai/sql-eval
[6] https://huggingface.co/datasets/b-mc2/sql-create-context
[7] https://yale-lily.github.io/spider
[8] https://bird-bench.github.io

At the time of writing, in terms of Execution with Values, the five tools listed on the Spider Leaderboard are (in descending order): "MiniSeek" (no reference available at the time of writing), "DAIL-SQL + GPT-4 + Self-Consistency" and "DAIL-SQL + GPT-4", both reported in (GAO et al., 2023), "DPG-SQL + GPT-4 + Self-Correction" (no reference available at the time of writing), "DIN-SQL + GPT-4" (POURREZA; RAFIEI, 2023); for later reference, the seventh is "C3 + ChatGPT + Zero-Shot" (DONG et al., 2023). The 5 top tools in the BIRD Leaderboard, in terms of Execution Accuracy, in turn, are "SFT CodeS-15B" and "SFT CodeS-7B" (no reference available for both tools), "DAIL-SQL + GPT-4", "DIN-SQL + GPT-4", and GPT-4.

The Awesome Text2SQL Web site[9] 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[10] lists other AI tools for text-to-SQL.

MAC-SQL (WANG et al., 2024) is a recent addition and features an LLM-based multi-agent collaborative framework, with a core decomposer agent for text-to-SQL generation with few-shot chain-of-thought reasoning and two auxiliary agents that use external tools to acquire smaller sub-databases and refine erroneous SQL queries.

Finally, LangChain[11] is a framework that helps develop LLM applications. LangChain is compatible with MySQL, PostgreSQL, Oracle SQL, Databricks, SQLite, and other DBMSs. Very briefly, *LangChain SQLQueryChain* extracts metadata from the database automatically, creates a prompt, and passes this metadata to the LLM. In particular, SQLQueryChain passes a view specification as if it were a table specification. This chain greatly simplifies creating prompts to access databases. In addition to passing the schema in the prompt, this chain makes it possible to provide sample data that can help an LLM build correct queries when the data format is not apparent. Sample rows are added to the prompt after the column information for each corresponding table

[9] https://github.com/eosphoros-ai/Awesome-Text2SQL
[10] https://topai.tools/s/Text-to-SQL-bot
[11] <https://docs.langchain.com>

2.5 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 (GAO et al., 2023). 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[12] 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[13] 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 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 (POURREZA; RAFIEI, 2024) 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 (JIANG et al., 2023) and DeepSeek-7B (DEEPSEEK-AI et al., 2024) 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 achieved a performance comparable to the best strategies listed in the Spider leaderboard. Although promising, these results reflect the limitations of Spider, namely, that the databases have simple schemas.

Natural-SQL-7B by ChatDB[14] 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[15] 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-text2-sql[16] had poor performance, even using RAG (see Section

[12]https://github.com/defog-ai/sqlcoder?tab=readme-ov-file
[13]https://defog.ai/blog/open-sourcing-sqlcoder/
[14]https://huggingface.co/chatdb/natural-sql-7b-GGUF
[15]https://github.com/cfahlgren1/natural-sql
[16]https://huggingface.co/support-pvelocity/Code-Llama-2-13B-instruct-text2sql-GGUF

2.6). This may reflect the fact that Code Llama 2-13B-instruct-text2-sql 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.

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

Retrieval-Augmented Generation (RAG), introduced in (LEWIS et al., 2020), 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.

There is an extensive literature on RAG. A recent survey (GAO et al., 2024) classified RAG strategies into *naive*, *advanced*, and *modular* RAG. Advanced RAG introduces various methods to optimize retrieval. Modular RAG integrates strategies to enhance functional modules, such as incorporating a search module for similarity retrieval and applying a fine-tuning approach in the retriever.

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 (GUO et al., 2023). A RAG technique is used in (PANDA; GOZLUKLU, 28 Feb 2024) 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.

In summary, the background section has provided a comprehensive overview of the evolution of Natural Language Processing techniques, particularly focusing on the challenges and advancements in Text-to-SQL systems. The exploration of Generative AI Language Models (GLMs), various datasets, and current Text-to-SQL tools has set the stage for understanding the complexities involved in translating natural language queries into SQL. Moreover, the detailed discussion on the Retrieval-Augmented Generation (RAG) technique has highlighted its potential to enhance the accuracy and relevance of Text-to-SQL outputs. With this foundation, we can now transition to our proposed methodology. The following chapter will present a detailed proposal of a RAG-based Text-to-SQL framework, including the generation of synthetic datasets and the incorporation of schema information, to address the challenges identified in the background research.

3 Proposal

3.1 Outline of a Prototype Text-to-SQL framework

This section outlines a prototype text-to-SQL framework based on SQL-QueryChain, combined with a Retrieval-Augmented Generation (RAG) technique, leaving as flexibilization points: the *GLM* and the database *DB*. Figure 3.1 depicts the framework, with the flexibilization points indicated in light grey.

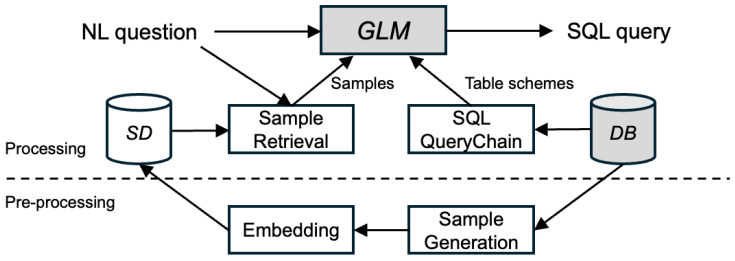

Figure 3.1: The architecture of a prototype text-to-SQL framework.

The simplest way to create a GLM-based text-to-SQL tool would be to design Prompts that implement three steps:

S1. (Schema Linking) Given an NL question Q, retrieve a description of the relation tables and columns, creating a context C.

S2. (SQL Translation) Ask to translate Q into an SQL query S under the context C.

S3. (SQL Execution) Execute S.

The text-to-SQL 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) Ask the GLM to retrieve from DB a description of the relation tables and columns, passing the context C; 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.

S4. (SQL Execution) Execute S_0.

SQLQueryChain[1], adopted in this study, 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 (COELHO et al., 2024; NASCIMENTO et al., 2024a), 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 a GLM, and led to a considerable improvement of the accuracy of text-to-SQL Prompt strategies (COELHO et al., 2024).

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 (COELHO et al., 2024) 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 4.3, discussed in Section 4.1.2, shows a sample of the synthetic dataset for Mondial.

This basic RAG technique has a limitation, though, since the Retrieval Step often retrieves 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 subquestions $Q_1, ..., Q_n$, which are then used in the Sample Retrieval step.

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

This study then introduces a revised version of the Sample Retrieval step:

S0. (Question Decomposition) Given an NL question Q_0, decompose Q_0 into subquestions $Q_1, ..., Q_n$.

S1. (Sample Retrieval) Embed $Q_0, Q_1, ..., Q_n$ into vectors $V_{Q_0}, V_{Q_1}, V_{Q_n}$; retrieve from DS the top-k vectors $V_{j,i}$, $j \in [0, n]$, $i \in [1, k]$, most similar to V_{Q_j}; create a context C with the pairs $(Q_{j,i}, S_{j,i})$, $j \in [0, n]$, $i \in [1, k]$.

3.2 The Proposed RAG-Based Technique

The RAG-based techniques assume that the NL questions in the synthetic dataset have already been embedded into a vector space and indexed accordingly. The critical step, *question similarity selection*, first obtains an embedding E_I of the input NL question. Then, it retrieves from the synthetic dataset the top-k pairs whose NL question embeddings are similar to E_I, as usual.

The experiments will consider four configurations, which test two synthetic datasets combined or not with schema information.

The *single-attribute synthetic dataset* is generated by sampling just single attributes (that is, by calling Algorithm 1 always with $n = 1$), whereas the *multi-attribute synthetic dataset* is generated by sampling multiple attributes. Therefore, the single-attribute syntactic dataset will contain only pairs (Q_N, Q_S), where Q_N is a *simple NL question* and Q_S is a SQL query over a single table, with no join clauses, and a `WHERE` clause with one filter over a single column.

Now, *RAG with no schema information* uses RAG to retrieve examples from the synthetic dataset which are similar to the input NL question, and prompts the LLM only with the retrieved examples. These experiments test whether the synthetic dataset is sufficient to convey all the information about the database the LLM requires for the text-to-SQL task.

By contrast, *RAG with schema information* uses RAG to retrieve examples from the synthetic dataset and prompts the LLM with the retrieved examples and the database schema. These experiments test whether the RAG-based technique adds information not conveyed by the schema, thereby leading to a better text-to-SQL prompt strategy.

3.3 Generation of the Synthetic Dataset

A *synthetic dataset* may 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. This section outlines a procedure to generate a synthetic dataset with examples of all these three types, by exploring the database, its schema, and associated documentation.

Algorithm 1 shows a much simplified pseudo-code of the core procedure, which generates a pair (Q_N, Q_S), where Q_N is a NL question and Q_S is the corresponding SQL query. Very briefly, the core procedure goes as follows:

Algorithm 1: Generate Synthetic Dataset

Input: the number n of columns to select, the database D_R, the database schema D_S, and the database documentation D_{doc}, if available.

Output: a pair (Q_N, Q_S) where Q_N is an NL question and Q_S is the corresponding SQL query.

```
1 Function GenerateExample(n, D_R, D_S, D_doc):
2     Q_A ← SelectColumns(n, D_S);
3     Q_K ← CreateNLQuestion(Q_A, D_R, D_S, D_doc);
4     Q_S ← CompileSQLQuery(Q_K, D_S);
5     Q_N ← ImproveNLQuestion(Q_K, D_R, D_S, D_doc);
6     return (Q_N, Q_S);
```

– Step 1 (on Line 2) selects a set Q_A of n pairs of table/column names from the database tables. The selection process employs a weighted random distribution, which reflects the likelihood of each column of a table being chosen by an average user.

– Step 2 (on Line 3) creates an NL question Q_K from Q_A by prompting GPT-3.5-turbo with the following information: the column/table pairs selected in Step 1, sample values of each column/table, and a simplified Data Definition Language (DDL) statement encompassing only the columns and tables involved, including any join tables. In addition, the type of restriction to be incorporated into the NL question depends on the nature of the data. For instance, for numerical columns, restrictions may involve operations such as summation, averaging, or finding the maximum value. Similarly, requests might include grouped aggregations for categorical columns, among other possibilities. Lastly, the prompt includes instructions on how to formulate the NL question by using the

database vocabulary without altering the column and table names. This is essential for the next step.

- Step 3 (on Line 4) calls GPT-4 to translate Q_K into an SQL query that responds to the NL question by providing the simplified DDL statement in the same manner as in Step 2. It is worth noting that, since Q_K is written using the database vocabulary and since the DDL statement describes only the necessary tables and columns, this translation task is relatively simple.
- Finally, Step 4 (on Line 5) calls GPT-3.5-turbo to translate Q_K into an improved NL question Q_N, using the database documentation D_{doc}, which includes the Description of each column and table along with synonyms. During this step, GPT-3.5-turbo is instructed to rephrase the NL question by translating from the database to the user's vocabulary, preserving the original NL question intent.

By looping Algorithm 1 over different combinations of table/column name samples, one can generate a reasonably large dataset, containing thousands of instances of NL questions and their corresponding SQL queries.

When $n = 1$, Algorithm 1 samples just one column/table pair from the database schema. This option is interesting for capturing data value semantics, as the following example illustrates (see Chapter 4 for a description of the Mondial dataset)

- Suppose Step 1 selects column `Area` of table `MONDIAL_LAKE`.
- Since `Area` is a categorical column, suppose Step 2 decides to create the filter `Area > 500`, based on the samples provided in the prompt. The NL question will then be $Q_K =$ "*List all instances on the Lake table which have Area greater than 500*".
- Step 3 creates the SQL query Q_S

Code 1: SQL query Q_S

```
1    SELECT * FROM MONDIAL_LAKE WHERE AREA > 500;
```

- Step 4 observes in the database documentation that users adopt *Big Lakes* to refer to plans such that `Area = 500`. Therefore, Step 4 generates the improved NL question $Q_N =$ "*List all big lakes*".

When $n > 1$, Algorithm 1 samples two or more column/table pairs from the database schema, which forces Step 3 to generate SQL queries with one or more joins, if the sampled column/table pairs are from different tables. For example, consider a sample with two column/table pairs (that is, $n = 2$):

- Suppose Step 1 selects column `ESTABLISHED` of table `Mondial_Organization` and column `ELEVATION` of table `Mondial_City`.
- Based again on the samples provided and the nature of the columns, suppose that Step 2 generates the following question: $Q_K =$ "*What types* ***ESTABLISHED*** *instances from the table* ***Mondial_Organization*** *that are equal to XPTO associated with* ***ELEVATION*** *equals XPTO from table* ***Mondial_City****.*
- Step 3 creates the SQL query Q_S

```
1    SELECT MO.ESTABLISHED
2    FROM MONDIAL_ORGANIZATION MO
3    JOIN MONDIAL_CITY MC
4    ON MO.CITY = MC.NAME
5    WHERE MC.ELEVATION = '%xpto';
```

- Step 4 then improves the readability of Q_K, generating the NL question $Q_N =$ "*What is the established for organizations in the organization table where the corresponding cities in the city table have a elevation equals xpto?*".

Finally, we observe that the implementation of the core procedure is capable of generating far more complex NL question/SQL query pairs. Without going into the details, the following example illustrates this remark:

- Initial NL question:"*What is the time period when organizations were established in cities with known elevation values?*"
- Reformulated NL question: "*time period organizations established cities known elevation values*"
- SQL query for both NL questions:

```
1    SELECT MIN(MO.ESTABLISHED),
2    MAX(MO.ESTABLISHED)
3    FROM MONDIAL_ORGANIZATION MO
4    JOIN MONDIAL_CITY MC
5    ON MO.CITY = MC.NAME
6    WHERE MC.ELEVATION IS NOT NULL;
```

4 Results

The experiments tested the strategies summarized in Table 4.6 over the Mondial benchmark. 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 positives and false negatives

4.1 The Mondial Benchmark

4.1.1 The Mondial Dataset and the Set of NL Questions

A *benchmark dataset* for the text-to-SQL task is a pair $B = (D, \{(L_i, G_i)/i = 1, ..., n\})$, where D is a database and, for $i = 1, ..., n$, L_i is an NL question over D and G_i is the *ground truth* SQL query over D that translates L_i. In the context of this study, a benchmark dataset is meant exclusively for testing text-to-SQL tools; it is not designed for training such tools. Section 4.3 describes the procedure adopted to evaluate text-to-SQL tools.

The primary benchmark dataset adopted in this study is Mondial, with a set of 100 NL questions and their translations to SQL.

The questions are classified into *simple*, *medium*, and *complex*, that correspond to the easy, medium, and hard classes used in the Spider benchmark (extra-hard questions were not considered). As in the Spider benchmark, the difficulty is based on the number of SQL constructs, so that queries that contain more SQL constructs (GROUP BY, ORDER BY, INTERSECT, nested subqueries, column selections, and aggregators) are considered to be harder. The list of questions contains 33 simple, 34 medium, and 33 complex questions[1].

Mondial stores geographic data and is openly available. It has a total of 47.699 instances; the relational schema[2] has 46 tables, with a total of 184 columns, and 49 foreign keys, some of which are multi-column.

The Mondial dataset provides a rich and diverse set of geographic data, encompassing information such as countries, cities, and rivers. This extensive

[1]The details are available at <https://github.com/dudursn/text_to_sql_chatgpt_real_world>

[2]The Mondial referential dependencies diagram can be found at <https://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-abh.pdf>

dataset is valuable for evaluating the performance of natural language to SQL translation systems due to its complexity and the variety of relationships it embodies. The dataset's schema, which includes 46 tables, ensures a broad spectrum of query types, making it an excellent choice for benchmarking purposes. The inclusion of multiple foreign keys, some of which span multiple columns, adds another layer of complexity, challenging systems to accurately interpret and navigate these relationships.

In addition to the structural richness of the Mondial dataset, the classification of questions into simple, medium, and complex categories provides a granular evaluation framework. Simple questions typically involve straightforward selections from single tables, while medium questions might include basic joins or filters. Complex questions, on the other hand, often require nested queries, aggregations, or advanced SQL constructs like GROUP BY and ORDER BY. This classification allows for a nuanced assessment of a system's capabilities, helping to pinpoint strengths and areas needing improvement across different levels of query complexity.

Table 4.1 shows basic statistics of the sets of queries, where "#cols" refers to the number of columns of the target clause and the other columns refer to the number of joins, filters, and aggregations that occur anywhere in the query, including any nested query.

Table 4.1: Datasets Statistics.

		Mondial			
	Type	**#cols.**	**#joins**	**#filters**	**#aggr**
Average	complex	1,09	1,41	1,65	0,38
	medium	1,18	0,79	0,85	0,30
	simple	1,33	0,42	0,73	0,12
Maximum	complex	2	4	5	1
	medium	2	3	1	2
	simple	4	3	2	1

Finally, Table 4.2 shows some samples of NL questions and their gold-standard SQL translations

Table 4.2: Sample benchmark NL questions and their SQL translations

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	What are the area, elevation, and type of lakes in Italy?	`SELECT l.AREA, l.ELEVATION, l.TYPE FROM MONDIAL_GEO_LAKE gl, MONDIAL_LAKE l, MONDIAL_COUNTRY c WHERE (gl.LAKE = l.NAME) AND (gl.COUNTRY = c.CODE) AND c.NAME = "Italy"`	59

4.1.2 The Mondial Synthetic Dataset

The Mondial benchmark also includes a synthetic dataset with 60,000 pairs, obtained by sampling table columns and associated data, using the technique outlined in Chapter 3.

As an example, Table 4.3 shows a sample of the synthetic dataset for Mondial. Lines 1 to 3 correspond to the sampled pair ("`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).

Lines 4 to 6 show more elaborate examples that correspond to 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 NL question paraphrases in Lines 4 to 6.

Table 4.4 presents an example of an NL question, the corresponding gold-standard SQL query, and the SQL queries generated with and without question decomposition. The example illustrates that the GLM tries to relate the country name "thailand" directly with the `COUNTRY` column of the `MONDIAL_RELIGION` table. However, the values of the `COUNTRY` column consist of the Country code, requiring a join with the `MONDIAL_COUNTRY` table to obtain the Country name. This behavior reflects the fact that the samples retrieved do not include examples of such joins, as shown in Line 1 of Table 4.5. However, by running the Question Decomposition step first, the Sample Retrieval step will fetch the examples in Lines 2 and 3 of Table 4.5. Note that Line 2 indeed contains an example of a join between the two tables, needed to answer the original NL question.

Table 4.5 illustrates how the Question Decomposition step works. The

Table 4.3: A sample of the synthetic dataset for Mondial.

#	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`

4	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`

5	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`

6	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`

original NL question (Line 1) requests the percentage of Hindu people. Thus, to answer this question, the Question Decomposition step generated the NL question on Line 2 to retrieve the total number of religious people, and the NL question on Line 3 to retrieve the number of Hindu people. By combining the results of these two subquestions, it would be possible to compute the percentage of Hindu people, as requested in the original NL question.

Table 4.4: Comparison of SQL generated with and without decomposition

NL Question	Gold-standard SQL Query	Generated SQL Query without Decomposition	Generated SQL Query with Decomposition

What is the percentage of religious people are hindu in thailand?	`SELECT r.percentage`
	`FROM mondial_religion r`
	`INNER JOIN mondial_country c`
	`ON r.country = c.code`
	`WHERE c.name = 'Thailand'`
	`AND r.name LIKE '%Hindu%'`
		`SELECT PERCENTAGE`
		`FROM MONDIAL_RELIGION`
		`WHERE COUNTRY = 'Thailand'`
		`AND NAME = 'Hindu';`
			`SELECT PERCENTAGE`
			`FROM MONDIAL_RELIGION r`
			`INNER JOIN MONDIAL_COUNTRY c`
			`ON r.COUNTRY = c.CODE`
			`WHERE LOWER(r.NAME) = 'hindu'`
			`AND LOWER(c.NAME) = 'thailand'`

4.2 Configuration of the Experiments

The experiments tested the strategies summarized in Table 4.6 for the 100 NL questions and their translations to SQL over the Mondial, stored in Oracle. The foreign keys of Mondial were used.

Except for the SQLCoder, the experiments ran each strategy with two LLMs – GPT-3.5-turbo or GPT3.5-turbo-16k and GPT-4. Since both schemas are fairly large, some experiments had to use GPT3.5-turbo-16k, which allows 16k tokens. The experiments used the (paid) OpenAI API.k.

Table 4.5: Generated decompositions with the RAG results.

Subquestion	**Sample Retrieved Pairs** (Q, S)
What is the percentage of religious people are Hindu in Thailand?	Q : What are the percentage values of the religions in the table MONDIAL_RELIGION? S : `SELECT NAME, PERCENTAGE FROM MONDIAL_RELIGION;`

What is the total number of religious people in Thailand?	Q : What is the total number of religious followers in all countries situated on the island of Tortola, in terms of the percentage of the population that practices a religion?
	
	S : `SELECT SUM(MONDIAL_RELIGION.PERCENTAGE)`
	`FROM MONDIAL_RELIGION`
	`JOIN MONDIAL_COUNTRY`
	`ON MONDIAL_RELIGION.COUNTRY = MONDIAL_COUNTRY.CODE`
	`JOIN MONDIAL_AIRPORT`
	`ON MONDIAL_COUNTRY.NAME = MONDIAL_AIRPORT.COUNTRY`
	`JOIN MONDIAL_ISLAND`
	`ON MONDIAL_AIRPORT.ISLAND = MONDIAL_ISLAND.NAME`
	`WHERE LOWER(MONDIAL_ISLAND.NAME) = 'tortola'`

What is the number of Hindu people in Thailand?	Q : What is the number of people living in each country in the database? S : `SELECT NAME, POPULATION FROM MONDIAL_COUNTRY`

SQLCoder used the `sqlcoder-34b-alpha` model, with 34B parameters. For the experiments, owing to constraints inherent in the model, the Mondial DDL was transposed to the PostgreSQL syntax, facilitated by GPT-4 under human supervision. Then, the output comprised SQL queries formulated in PostgreSQL syntax, subsequently transcribed into the Oracle syntax through GPT-4, again under human supervision.

The experiments that used the text-to-SQL framework, described in Chapter 3, instantiated with the Mondial database, ran three versions of the framework, identified as:

- *SQLQueryChain*: just Steps S2, S3, and S4 of the framework, that is, the standard SQLQueryChain implemented in LangChain. This is the baseline strategy.
- *SQLQueryChain with RAG*: Steps S1 to S4, using the synthetic dataset with 60,000 pairs pre-computed for Mondial, defined in Section 4.1.1. 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, S3 and S4. These experiments assess whether Question Decomposition further improves accuracy for a given model, as proposed in this work.

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:

Table 4.6: Summary of the strategies tested.

1) **"SQLCoder"** – Defog's tool for text-to-SQL, using a special-purpose LLM.
2) **"SQLQueryChain (LangChain)"** – The LangChain SQLQueryChain processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
3) **"SQLDatabaseSequentialChain (LangChain)"** – The LangChain SQLDatabaseSequentialChain processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
4) **"SQL Database Agent (LangChain)"** – The LangChain SQL Database Agent processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
5) **"DIN-SQL"** – An implementation of "DIN-SQL" that allows the use of different LLMs; tested with GPT-3.5-turbo-16k and GPT-4.
6) **"C3"** – An implementation of "C3" that allows the use of different LLMs; tested with GPT-3.5-turbo and GPT-4.
7) **"C3+DIN"** – The C3+DIN tool introduced in Section 2.4; tested with GPT-3.5-turbo-16k and GPT-4.

- *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 (see Table 2.1).

4.3 Evaluation Procedure

Let $B = (D, \{(P_i, G_i)/i = 1, ..., n\})$ be a benchmark dataset. Let P_i be a predicted SQL query and G_i be the corresponding ground truth SQL query. Let PT_i and GT_i be the tables that P_i and G_i return when executed over D, called the *predicted* and the *ground truth* tables.

Intuitively, P_i is *correct* if PT_i and GT_i are similar. The notion of similarity adopted neither requires that PT_i and GT_i have exactly the same columns, nor that they have exactly the same rows. This allows for some mismatch between PT_i and GT_i. The following procedure captures this intuition:

1. Compute GT_i and PT_i over D.

2. For each column of GT_i, compute the most similar column of PT_i, respecting a minimum column similarity threshold of tc. This step induces a partial matching M from columns of GT_i to columns of PT_i.

3. If the fraction of the number of columns of GT_i that match some column of PT_i is below a given threshold tn, P_i is considered *incorrect*.

4. The *adjusted ground truth table* AGT_i is constructed by dropping all columns of GT_i that do not match any column of PT_i, and the *adjusted predicted table* APT_i is constructed by dropping all columns of PT_i that are not matched and permuting the remaining columns so that PC_k is the k^{th} column of APT_i if GC_k, the k^{th} column of AGT_i, is such that $M(GC_k) = PC_k$.

5. Finally, AGT_i and APT_i are compared. If their similarity is above a given threshold tq, then P_i is *correct*; otherwise P_i is *incorrect*.

In Step 1, GT_i may be pre-computed to avoid re-executing G_i over D for each experiment.

In Step 2, the similarity between two table columns was measured as their Jaccard coeficient (recall that table columns are sets). The threshold tc was set to 0.50.

In Step 3, the threshold tn was set to 0.80, that is, 0.80 of the number of columns of GT_i must match some column of PT_i. Note that, setting $tn = 0.80$ forces all columns of GT_i to match some column of PT_i, if GT_i has 4 or less columns (indeed $4 * 0.80 = 3.20$ is rounded up to 4, that is, GT_i must have all 4 columns matching some column of PT_i, and likewise for a smaller number of columns).

Now, recall from column "#cols" of Table 4.1 that all queries for Mondial have a result with at most 4 columns. Hence, setting $tn = 0.80$ implies that actually all columns of GT_i must match a column of PT_i.

In Step 4, the new tables AGT_i and APT_i will have the same number of columns and the matched columns will appear in the same order.

In Step 5, the similarity of AGT_i and APT_i was computed as their Jaccard coeficient (recall that tables are sets of tuples), and the threshold tq was set

to 0.95. Thus, AGT_i and APT_i need not have exactly the same rows but, intuitively, P_i will be incorrect if APT_i contains only a small subset of the rows in AGT_i, or APT_i contains many rows not in AGT_i.

Finally, the *accuracy* of a given text-to-SQL strategy over the benchmark B is the number of correct predicted SQL queries divided by the total number of predicted queries, as usual.

4.4 Prompt Strategies Results

Table 4.8, show the results for the Mondial database. Columns under "**Accuracy**" indicate the accuracy results for the simple, medium and complex queries, as well as the overall accuracy; columns "**Input Tokens**" and "**Output Tokens**" respectively show the number of tokens passed as input and received as output from the model; column "**Estim. Cost**" indicates the estimated cost in US Dollars; and column "**Exec. Time**" displays the total time to compute the 100 queries, which naturally depends on the HW and SW setup, and should be used only to compare the various strategies.

Table 4.7: Error analysis of the C3 with GPT-4 experiment using the Mondial database.

Error Type	Schema Linking	Joins	Nested Query	Invalid Query	Misc
Percentage	53.3	30.0	3.3	6.7	6.7

Accuracy. The top-5 strategies for overall accuracy used GPT-4. C3 had the best overall accuracy of 0.78. Then, SQLQueryChain with samples, DIN, and C3+DIN had the same overall accuracy of 0.70. Lastly, SQLQueryChain, without samples, achieved 0.69. Among the Langchain-based strategies, those that passed the entire schema and sample data in the prompt proved to have a superior overall accuracy. SQLCoder had a limited overall accuracy of 0.35. C3 with GPT-4 also had the best accuracy for complex queries, 0.71, whereas C3+DIN with GPT-4 had the best accuracy for simple queries, 0.91.

Strategy details. Experiments with Langchain were divided into two groups: passing the NL question and the schema; passing the NL question, the schema, and two sample rows from each table. The second group resulted in large prompts, requiring GPT-3.5-turbo-16k in some cases, while GPT-4 handled large prompts seamlessly. SQLDatabaseSequentialChain and SQLAgent had minimal cost due to the smaller prompts obtained by filtering schemas for tables. However, ChatGPT misidentified crucial tables, leading to incorrect SQL queries. SQLAgent became lost or hallucinated using GPT-4. In fact,

SQLAgent is not fully compatible with GPT-4. Also, SQLAgent had a poor performance with GPT-3.5-turbo.

DIN with GPT-3.5-turbo had the largest number of input tokens, followed closely by DIN with GPT-4, both with over 1,4 MM input tokens. Indeed, DIN generates large prompts since it passes the complete database schema and uses a few examples to indicate how the LLM should reason and generate SQL code in each stage, except for the self-correction stage.

C3's Consistent Output generated many output tokens since it produces ten answers in each clear-prompting stage (table recall and column recall). Furthermore, the output token price of ChatGPT ($0.06/1K sampled tokens) is higher than the input token price ($0.03/1K prompt tokens). Thus, albeit C3 with GPT-4 had the best overall accuracy, it generated 426,937 output tokens with an overall cost of $30.23.

Table 4.7 shows the error analysis of C3 with GPT-4. When compared with that of C3 for the Spider benchmark (DONG et al., 2023), it indicates that the errors resulting from schema linking and joins are exacerbated in the experiments with Mondial, which would be expected, given that the Mondial schema is far more complex than the majority of the datasets in the Spider benchmark.

C3+Din with GPT-4 had the same overall accuracy as DIN, but lower than C3 with GPT-4. However, its cost and execution times were the highest among all experiments. It inherited the problems of C3 and DIN, such as a high table and column recall time and large prompt sizes, but generated fewer input tokens than DIN, as it did not use all DIN modules.

Table 4.8: Results for Mondial.

#Line	Model		#Corrrect Predicted Queries				Accuracy									
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Total	Input Tkn	Output Tkn	Tokens	Est. Cost	Exec Time	Comm.
1	***SQLCoder***		15	13	7	35	0.45	0.39	0.21	0.35	---	---	---	---		
	SQLQueryChain (LangChain)															
2	GPT-3.5-turbo		0	0	0	55	0.76	0.58	0.32	0.55	569,713	2,563	572,276	$1.72	0:02:15	
3	GPT-4		0	0	0	69	0.82	0.79	0.47	0.69	569,713	2,643	572,356	$17.25	0:05:28	
	SQLQueryChain (LangChain) - with samples															
4	GPT-3.5-turbo-16k		0	0	0	60	0.76	0.67	0.38	0.60	782,813	2,501	785,314	$2.36	0:06:47	(1)(2)(3)
5	GPT-4		0	0	0	70	0.82	***0.85***	0.44	0.70	782,624	2,527	785,151	$23.63	0:09:20	(2)
	SQLDatabaseSequentialChain (LangChain)															
6	GPT-3.5-turbo		0	0	0	43	0.67	0.39	0.24	0.43	63,394	2,883	66,277	$0.10	0:23:01	
7	GPT-4		0	0	0	63	0.67	0.67	0.56	0.63	67,645	3,276	70,921	$2.23	0:06:58	
	SQLDatabaseSequentialChain (LangChain) - with samples															
8	GPT-3.5-turbo		0	0	0	40	0.67	0.39	0.15	0.40	73,994	2,955	76,949	$0.12	0:02:56	(2)
9	GPT-4		0	0	0	59	0.67	0.61	0.50	0.59	80,208	3,217	83,425	$2.60	0:07:00	(2)
	SQL Database Agent (LangChain)															
10	GPT-3.5-turbo		0	0	0	37	0.58	0.33	0.21	0.37	376,236	16,898	393,134	$0.60	0:21:35	
11	GPT-4		---	---	---	---	---	---	---	---	---	---	---	---	---	(4)
	SQL Database Agent (LangChain) - with samples															
12	GPT-3.5-turbo		0	0	0	35	0.58	0.36	0.12	0.35	441,785	17,939	459,724	$0.70	0:32:21	(2)
13	GPT-4		---	---	---	---	---	---	---	---	---	---	---	---	---	(4)
	DIN															
14	GPT-3.5-turbo-16k		0	0	0	56	0.82	0.45	0.41	0.56	1,431,645	33,050	1,464,695	$4.43	0:14:54	(1)(5)
15	GPT-4		0	0	0	70	0.85	0.76	0.50	0.70	1,428,284	32,473	1,460,757	$44.80	0:45:26	
	C3															
15	GPT-3.5-turbo		0	0	0	55	0.79	0.48	0.38	0.55	175,298	754,619	929,917	$1.77	1:09:27	(6)
16	GPT-4		0	0	0	78	0.88	0.76	***0.71***	***0.78***	153,705	426,937	580,642	$30.23	1:20:06	
	C3+DIN															
17	GPT-3.5-turbo-16k		0	0	0	59	0.82	0.52	0.44	0.59	1,147,061	719,891	1,866,952	$6.32	1:16:52	(1)(5)(6)
18	GPT-4		0	0	0	70	***0.91***	0.70	0.50	0.70	1,137,152	401,191	1,538,343	$58.19	1:48:36	(6)

Notes:
(1) gpt-3.5 turbo-16k was used due to its larger token limit.
(2) Two samples were passed in the prompt.
(3) The complete schema could be passed in the prompt.
(4) GPT-4 is not entirely compatible with SQLAgent.
(5) The DIN prompt requires a larger token limit.
(6) Table and column metadata recall took a long time.

4.5 RAG Framework Results

Recall that the Mondial benchmark has 100 NL questions, where 33 are simple, 33 are medium, and 34 are complex. Table 4.9 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 RAG alternative.

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 Section 4.4, 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, it should be stressed that 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 had good performance on a coding task (see again Table 2.1).

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 described in Chapter 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 either changing

the column name to `COUNTRY_CODE` or, better still, by creating a view that includes a predefined column, `COUNTRY_NAME`, such as

```
MONDIAL_CITY_VIEW(NAME, COUNTRY_NAME, COUNTRY_CODE, PROVINCE, POPULATION, LATITUDE,
                  LONGITUDE, ELEVATION)
```

This simple solution proved quite effective for a real-world private database with a large schema that was not considered *LLM-friendly* (NASCIMENTO et al., 2024b).

Table 4.9: 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	26	16	69	0,82	0,79	0,47	0,69
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

5 Conclusion and future work

This work 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 large, complex schemas typical of real-world applications. The work 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 large, complex 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 (COELHO et al., 2024; NASCIMENTO et al., 2024a).

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 mostly simple.

The results reported in the work 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 Chapter 3 and Section 4.1.2, 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 4.5.

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 work 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, other real-world databases (with questions) could be added to strengthen testing the models.

6 Bibliography

COELHO, G. et al. Improving the accuracy of text-to-sql tools based on large language models for real-world relational databases. In: **(Accepted to the 35th International Conference on Database and Expert Systems Applications (DEXA 2024))**. [S.l.: s.n.], 2024. Cited 2 times in pages 24 and 41.

DATABRICKS. **Introducing DBRX: A New State-of-the-Art Open LLM**. 2024. Https://platform.openai.com/docs/models/overview. Cited in page 16.

DEEPSEEK-AI et al. **DeepSeek LLM: Scaling Open-Source Language Models with Longtermism**. 2024. Https://doi.org/10.48550/arXiv.2401.02954. Cited in page 21.

DONG, X. et al. **C3: Zero-shot Text-to-SQL with ChatGPT**. 2023. Cited 3 times in pages 19, 20, and 37.

GAN, Y. et al. Towards robustness of text-to-sql models against synonym substitution. **CoRR**, abs/2106.01065, 2021. Https://doi.org/10.48550/arXiv.2106.01065. Cited in page 18.

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**. [S.l.: s.n.], 2021. p. 8926–8931. Https://doi.org/10.18653/v1/2021.emnlp-main.702. Cited in page 18.

GAO, D. et al. **Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation**. 2023. Cited 3 times in pages 19, 20, and 21.

GAO, Y. et al. Retrieval-augmented generation for large language models: A survey. **arXiv preprint**, 2024. Https://doi.org/10.48550/arXiv.2312.10997. Cited in page 22.

GUO, C. et al. Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain. **arXiv preprint**, 2023. Https://doi.org/10.48550/arXiv.2307.05074. Cited in page 22.

JIANG, A. Q. et al. Mistral 7b. **arXiv preprint**, 2023. Https://doi.org/10.48550/arXiv.2310.06825. Cited in page 21.

JURAFSKY, D.; MARTIN, J. H. **Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition**. [S.l.]: Prentice Hall, 2023. Third Edition draft. Cited in page 15.

LEWIS, P. et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. In: LAROCHELLE, H. et al. (Ed.). **Advances in Neural Information Processing Systems**. [S.l.]: Curran Associates, Inc., 2020. v. 33, p. 9459–9474. Https://api.semanticscholar.org/CorpusID:218869575. Cited in page 22.

LI, J. et al. **Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs**. 2023. Cited in page 18.

MANNING, C. D. Human Language Understanding & Reasoning. **Daedalus**, v. 151, n. 2, p. 127–138, 05 2022. ISSN 0011-5266. Doi: 10.1162/daed_a_01905. Disponível em: <https://doi.org/10.1162/daed_a_01905>. Cited in page 15.

NASCIMENTO, E. R. et al. Text-to-sql meets the real-world. In: **Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1**. [S.l.: s.n.], 2024. p. 61–72. Cited 2 times in pages 24 and 41.

NASCIMENTO, E. R. et al. My database user is a large language model. In: **Proceedings of the 26th International Conference on Enterprise Information Systems – Volume 1**. [S.l.: s.n.], 2024. p. 800–806. Cited 2 times in pages 19 and 40.

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. Cited in page 22.

PING, W. J. **Open-sourcing SQLEval: our framework for evaluating LLM-generated SQL**. 2023. Disponível em: <https://defog.ai/blog/open-sourcing-sqleval/>. Cited in page 19.

POURREZA, M.; RAFIEI, D. **DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction**. 2023. Cited 2 times in pages 19 and 20.

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. Cited in page 21.

TUNSTALL, L.; WERRA, L. von; WOLF, T. **Natural Language Processing with Transformers, Revised Edition**. [S.l.]: O'Reilly Media, Inc., 2022. ISBN 9781098136796. Cited in page 15.

VASWANI, A. et al. Attention is all you need. In: **Proceedings of the Advances in Neural Information Processing Systems 30 (NIPS'17)**. [S.l.: s.n.], 2017. v. 30. Cited in page 15.

WANG, B. et al. Mac-sql: A multi-agent collaborative framework for text-to-sql. **arXiv preprint**, 2024. Https://doi.org/10.48550/arXiv.2312.11242. Cited in page 20.

YU, T. et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In: RILOFF, E. et al. (Ed.). **Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing**. Brussels, Belgium: Association for Computational Linguistics, 2018. p. 3911–3921. Disponível em: <https://aclanthology.org/D18-1425>. Cited in page 18.

ZHONG, V.; XIONG, C.; SOCHER, R. Seq2sql: Generating structured queries from natural language using reinforcement learning. **CoRR**, abs/1709.00103, 2017. Disponível em: <http://arxiv.org/abs/1709.00103>. Cited in page 19.