Improving the Accuracy of Text-to-SQL Tools based on Large Language Models for Real-World Relational Databases

Gustavo M.C. Coelho[1 [0000−0003−2951−4972]],
Eduardo R.S. Nascimento[1 [0009−0005−3391−7813]],
Yenier T. Izquierdo[1 [0000−0003−0971−8572]],
Grettel M. García[1 [0000−0001−9713−300X]],
Lucas Feijó[1 [0009−0006−4763−8564]],
Melissa Lemos[1 [0000−0003−1723−9897]],
Robinson L.S. Garcia[2 [0000−0002−0528−5151]],
Aiko R. de Oliveira[1,3 [0009−0001−8970−0454]],
João P. Pinheiro[3 [0000−0002−0909−4432]], and
Marco A. Casanova[1,3 [0000−0003−0765−9636]]

[1] Instituto Tecgraf, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil
{gustavocoelho,rogerrsn,ytorres,ggarcia,lucasfeijo,melissa}
@tecgraf.puc-rio.br
[2] Petrobras, Rio de Janeiro, 20031-912, RJ, Brazil
robinson.garcia@petrobras.com.br
[3] Departamento de Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil
{casanova,aramalho,jpinheiro}@inf.puc-rio.br

Abstract. Real-world relational databases (RW-RDB) have large, complex schemas often expressed in terms alien to end-users. This scenario is challenging to LLM-based text-to-SQL tools, that is, tools that translate Natural Language (NL) sentences into SQL queries using a Large Language Model (LLM). Indeed, their accuracy on RW-RDBs is considerably less than that reported for well-known synthetic benchmarks. This paper then introduces a technique to improve the accuracy of LLM-based text-to-SQL tools on RW-RDBs using Retrieval-Augmented Generation. The technique consists of two steps. Using the RW-RDB schema, the first step generates a synthetic dataset E of pairs (Q_N, Q_S), where Q_N is an NL sentence and Q_S is the corresponding SQL translation. The core contribution of the paper is an algorithm that implements this first step. Given an input NL sentence Q_I, the second step retrieves pairs (Q_N, Q_S) from E based on the similarity of Q_I and Q_N, and prompts such pairs to the LLM to improve accuracy. To argue in favor of the proposed technique, the paper includes experiments with an RW-RDB, which is in production at an Energy company, and a well-known text-to-SQL prompt strategy. It repeats the experiments with Mondial, an openly available database with a large schema. These experiments constitute a second contribution of the paper.

Keywords: Text-to-SQL · Retrieval-Augmented Generation · RAG · GPT · Large Language Models · Relational Databases.

1 Introduction

Natural language (NL) interfaces to databases (NLIDBs) allow users to access databases using questions formulated in Natural Language (NL) [1]. An efficient way to construct an NLIDB is to adopt a *text-to-SQL tool* that translates NL sentences to SQL queries. A text-to-SQL tool is *generic* (or *cross-domain*) if it is designed to work with any database; by contrast, a tool is *database-specific* if it is constructed for a particular database.

A large number of generic text-to-SQL tools have been constructed with relative success [1, 6, 7] over well-known benchmarks [9, 14]. The leaderboards of these benchmarks indicate that the best tools are currently based on Large Language Models (LLMs) [10], that is, they are *LLM-based text-to-SQL tools*. However, when applied to real-world relational databases (RW-RDBs), the performance of such tools is significantly less than that reported on the leaderboards.

Indeed, RW-RDBs are challenging for at least four reasons:

1. The relational schema is often an inappropriate specification of the database from the point of view of the LLM – the table and column names are often different from the terms the users adopt to formulate their NL questions.
2. The database schema is often large, in the number of tables, columns per table, and foreign keys – but a large schema may not fit in the prompt area, and opens space to queries with many joins, which are difficult to synthesize.
3. The data semantics is often complex; for example, some data values may encode enumerated domains – again, the terms the users adopt to formulate their NL questions may have to be mapped to this internal semantics.
4. Metadata and data are often ambiguous, which influence the behavior of an LLM-based text-to-SQL tool, leading to unexpected results.

To address these challenges, Nascimento et al. [11] argued that the text-to-SQL task can be facilitated by providing a database specification based on *LLM-friendly views* that are close to the language of the users' questions and that eliminate frequently used joins, and *LLM-friendly data descriptions* of the data semantics. However, the LLM-friendly data descriptions were limited in [11] to passing, in the prompt, a few tuples of each table used to process the NL question, which is a weak solution to capture data semantics and solve ambiguities.

This paper then concentrates on the problem of constructing database-specific LLM-based text-to-SQL tools for RW-RDBs, defined as: "Given an RW-RDB D, construct an LLM-based text-to-SQL tool such that, given any NL question on D, the tool translates the NL question into an equivalent SQL query on D".

The paper introduces a RAG-based technique, that is, a technique based on Retrieval-Augmented Generation [8], that provides a robust strategy to construct database-specific text-to-SQL tools for RW-RDBs, especially when it comes to conveying the data semantics of the RW-RDB to the LLM. The RAG-based technique consists of two steps. The first step generates a *synthetic dataset* E of pairs (Q_N, Q_S), where Q_N is an NL sentence and Q_S is the corresponding SQL translation. It is based on a technique that samples the RW-RDB and its schema and associated documentation, and calls an LLM to create Q_N from the sampled

data and to translate Q_N into Q_S. Roughly, by varying how the sampling works, the pairs in E help address all four RW-RDB challenges by providing text-to-SQL examples, including pairs that expose data semantics. Therefore, the dataset E is specific to the RW-RDB, but the algorithm is generic and applicable to any RW-RDB. Given an input NL sentence Q_I, the second step retrieves samples (Q_N, Q_S) from E based on the similarity of Q_I and Q_N, and prompts such pairs to the LLM to improve accuracy. The core contribution of the paper is an algorithm to generate a synthetic dataset E from an RW-RDB and its schema and associated documentation.

To argue in favor of the RAG-based technique, the paper includes experiments with the same RW-RDB and the same set of questions as in [11], and a text-to-SQL prompt strategy, implemented with LangChain using GPT-3.5 and GPT-4, which includes the RAG-based technique. The results suggest that the RAG-based technique indeed leads to improved accuracy. Since the RW-RDB is proprietary, the paper repeats the experiments with Mondial, an openly available database with a large, complex relational schema. As the Mondial schema adopts a user-friendly vocabulary, using LLM-friendly views is not required. On these challenging databases, the proposed RAG-based technique achieved an accuracy close to that obtained by the best strategies on the (much simpler) benchmark databases. These experiments constitute a second contribution of the paper.

This paper is organized as follows. Section 2 covers related work. Section 3 summarizes the RW-RDB benchmark adopted and the LLM-friendly views solution. Section 4 details the RAG-based technique. Section 5 describes the experiments with the RW-RDB. Section 6 summarizes the experiments with Mondial. Finally, Section 7 contains the conclusions.

2 Related Work

Text-to-SQL Datasets. The Spider – Yale Semantic Parsing and Text-to-SQL Challenge [14] 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. The set of NL questions introduced in Section 3.3 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 tables or fewer. Also, all Spider NL questions are phrased in terms used in the database schemas. These two limitations considerably simplify the text-to-SQL task. Therefore, 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 databases.

Spider has two interesting variations. Spider-Syn [2] is used to test how well text-to-SQL tools handle synonym substitution, and Spider-DK [3] addresses testing how well text-to-SQL tools deal with domain knowledge.

BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation [9] 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. 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.

Despite the availability of these benchmarks for text-to-SQL, and inspired by them, Section 3 describes a benchmark tuned to the problem addressed in this paper. 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 [11], and a set of 100 test NL questions, that mimic those posed by real users, and their ground truth SQL translations.

Text-to-SQL Tools. The Spider Web site[4] publishes a leaderboard with the best-performing text-to-SQL tools. At the time of this writing, the top 5 tools achieved an accuracy that ranged 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 first prompt is, therefore, prone to failure if the database schema induces a vocabulary which is disconnected from the NL question terms. This failure cannot be easily fixed by even more elaborate prompts that try to match the schema and the NL question vocabularies, as argued in [11].

The BIRD Web site[5] also publishes a leaderboard. At the time of this writing, out of the top 5 tools, two use GPT-4, one uses CodeS-15B, one CodeS-7B, and one is not documented. The sixth and seventh tools also use GPT-4, appear in the Spider leaderboard, and are well-documented.

Finally, LangChain[6] is a generic framework that offers several predefined strategies to build and run SQL queries based on NL prompts. Section 5.1 uses LangChain to create a text-to-SQL prompt strategy.

Retrieval-Augmented Generation – RAG. Retrieval-Augmented Generation (RAG), introduced in [8], is a strategy to incorporate data from external sources before proceeding to the generation phase. 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. Recent references include a RAG technique for an LLM-based Text-to-SQL framework involving sample-aware prompting and a dynamic revision chain [5]. A RAG technique is used in [13] to retrieve the table and column descriptions to ensure that the NL question is related to the right tables and columns. A recent survey can be found in [4], encompassing the *Naive RAG*, the *Advanced RAG*, and the *Modular RAG*.

[4] https://yale-lily.github.io/spider
[5] https://bird-bench.github.io
[6] https://python.langchain.com

The core contribution of the paper is an algorithm that, given an RW-RDB D_R and 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 an RAG technique to improve the accuracy of a text-to-SQL strategy on D_R.

3 A Real-World Benchmark for the Text-to-SQL Task

This section describes a benchmark to help investigate the text-to-SQL task over an RW-RDB. It should be stressed that this benchmark was designed exclusively for testing text-to-SQL tools; it was not meant for training such tools.

3.1 The Real-World Relational Database

The RW-RDB is proprietary and stores data related to the integrity management of an energy company's industrial assets. The relational schema contains 27 relational tables with, in total, 585 columns and 30 foreign keys (some multi-column); the largest table has 81 columns.

Table and column names in the relational schema do not follow a specific vocabulary. This scenario implies that end-users have difficulty understanding the semantics of the stored data and must turn to database specialists, even if they have access to the database documentation.

But there is a second, more challenging problem: some column values are not end-user-friendly. Indeed, some column values, or combinations of column values, hide semantic information that does not directly correspond to end-user terms. To overcome this situation, database experts often create SQL functions that contain the logic to represent the semantics hidden in the column values. Still, end-users adopt their vocabulary to refer to these data values, such as "overdue order" which translates to "`Maintenance_Order.Status = 1`"; this translation is in the database documentation, but it is not readily visible.

3.2 The Sets of Views

To avoid the effect of the internal naming convention of the RW-RDB, the benchmark introduces three sets of LLM-friendly views of increasing complexity:

- *Conceptual schema views*: a set of views that define a one-to-one mapping of the relational schema to end users' terms; the views basically rename tables and columns.
- *Partially extended views*: a set of views that extend the conceptual schema views with new columns that predefine joins that follow foreign keys, as well as other selected columns.
- *Fully extended views*: a set of views such that each view combines several conceptual schema views; the set may optionally include some conceptual schema views.

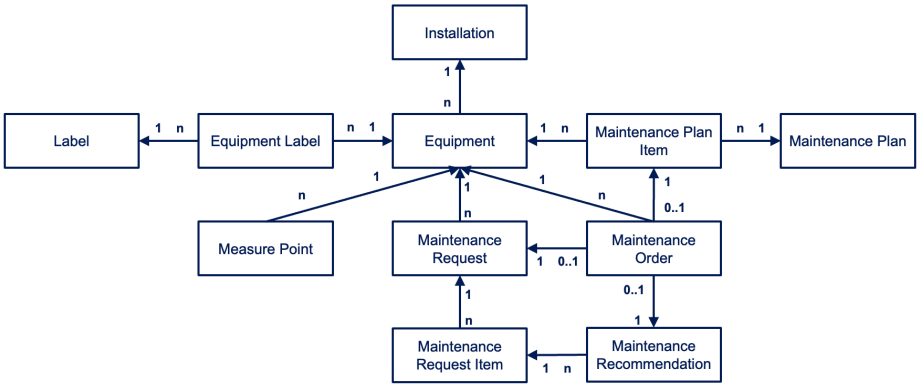

Fig. 1. The referential dependency diagram of a simplified version of the RW-RDB.

Table 1. A sample of the NL questions and their translations.

ID	NL Question	Ground Truth SQL Query Table	Question Type

1	Which IBX15 installation recommendations are expired?	`SELECT id`
		`FROM vw_maintenance_recommentation`
		`WHERE expired ='true' AND installation_name = 'IBX-15'`
			medium

…			

25	Show all orders with type MO05	`SELECT id`
		`FROM vw_maintenance_order`
		`WHERE type = 'MO05'`
			simple

…			

47	Which IBX-67 installation label has the most approved maintenance requests with a priority greater than 6?	`SELECT el.label_id`
		`FROM vw_maintenance_request mr`
		`JOIN vw_equipment_label el`
		`ON mr.equipment_id = el.equipment_id`
		`WHERE mr.installation_name = 'IBX-67'`
		`AND mr.situation = 'Approved' AND mr.priority > 6`
		`GROUP BY el.label_id`
		`ORDER BY COUNT(mr.id) DESC`
		`FETCH FIRST 1 ROWS ONLY`
			complex

…			

Figure 1 shows the referential dependencies diagram of a much-simplified version of the conceptual schema views, where an arrow represents a foreign key and points to the referenced table, as usual.

The set of *partially extended views* was defined by including the non-primary key columns of the view `Installation` into the views `Equipment`, `Maintenance_Order`, `Maintenance_Request`, `Maintenance_Recommendation`, and `Maintenance_Plan_Item`. The following statement shows the SQL code that creates the partially extended view `PE_Equipment` by combining the views `Installation` and `Equipment`:

```
CREATE VIEW pe_equipment AS
SELECT inst.name AS installation_name, inst.asset,
       inst.main_hub, inst.business_unit, equip.*
FROM equipment equip JOIN installation inst
      ON inst.id = equip.installation_id
```

where the `Installation` view has columns `id` (*primary key*), `name`, `asset`, `main_hub`, and `business_unit`, and the `installation_id` column of the `Equipment` view is a foreign key to the `id` column of `Installation`.

Finally, the set of fully extended views was defined following a similar strategy but combining two or more conceptual schema views. For instance, view `FE_Installation_Equipment_Maintenance_Request` combines views `Installation`, `Equipment`, and `Maintenance Request`.

3.3 The Test Questions and Their Ground Truth SQL Translations

The benchmark contains a set of 100 NL questions, $L = \{L_1, ..., L_{100}\}$, that consider the terms and questions experts use when requesting information related to the maintenance and integrity processes.

The ground truth SQL queries, $G = \{G_1, ..., G_{100}\}$, were manually defined over the conceptual schema views so that the execution of G_i returns the expected answer to the NL question L_i. The use of the conceptual schema views facilitated this manual task, since these views use a vocabulary close to that of the NL questions.

An NL question L_i is classified into *simple*, *medium*, and *complex*, based on the complexity of its ground truth SQL query G_i, as in the Spider benchmark (extra-hard questions were not considered). The set L contains 33 simple, 33 medium, and 34 complex questions.

Note that the NL question classification is anchored on the conceptual schema views. But, since these views map one-to-one to the tables of the relational schema, a classification anchored on the relational schema would remain the same. The classification is maintained for the other sets of views, even knowing that the definition of these other sets of views might simplify the translation of some NL questions (which was one of the reasons for considering these sets of views in the first place).

Table 1 shows some NL questions and their ground truth SQL translations.

4 The Proposed RAG-Based Technique

4.1 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: GenerateExample

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 $Q_A \leftarrow$ SelectColumns(n, D_S);
3 $Q_K \leftarrow$ CreateNLQuestion(Q_A, D_R, D_S, D_{doc});
4 $Q_S \leftarrow$ CompileSQLQuery(Q_K, D_S);
5 $Q_N \leftarrow$ 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:

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

```
SELECT * FROM Maintenance_Plan WHERE Classification = 1;
```

- Step 4 observes in the database documentation that users adopt *Critical Plans* to refer to plans such that `Classification = 1`. Therefore, Step 4 generates the improved NL question Q_N = "*List all critical plans*".

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 `Description` of table `Maintenance_Request` and column `Code_Name` of table `Installation`.
- Based again on the samples provided and the nature of the columns, suppose that Step 2 generates the following question: Q_K = "*List the `Code_Name` instances from the table `Installation` that are equal to XPTO associated with `Description` containing the word "paint" from table `Installation`.*
- Step 3 creates the SQL query Q_S

```
SELECT M.ID
FROM Maintenance_Request M
     JOIN Equipment E ON E.ID = M.Equipment_ID
     JOIN Installation I
          ON I.ID = E.Installation_ID
WHERE I.Code_Name = 'XPTO' AND
      LOWER(M.Description) LIKE '%paint%';
```

- Step 4 then improves the readability of Q_K, generating the NL question Q_N = "*List all paint maintenance requests for installation XPTO*".

Note that the SQL query Q_S contains two joins to navigate from table `Maintenance_Request` to table `Installation`, via table `Equipment`, as depicted in Figure 1. That is, relating column `Description` of table `Maintenance_Request` and column `Code_Name` of table `Installation` is somewhat more challenging.

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: "*How many maintenance orders from table `Maintenance_Order` are associated with each classification category from table `Maintenance_Plan`, considering only maintenance orders whose `Description` contains the word `'paint'`?*"

- Reformulated NL question: "*How many maintenance orders are associated with each maintenance plan classification, considering only maintenance orders whose Description contains the word 'paint'?*"
- SQL query for both NL questions:

```
SELECT P.Classification,
       COUNT(M.ID) AS Measument_Quantity
FROM Maintenance_Plan P
     JOIN Maintenance_Plan_Item I
          ON P.ID = I.Plan_ID
     JOIN Maintenance_Order M
          ON I.ID = M.Item_ID
WHERE LOWER(M.Description) LIKE '%paint%'
GROUP BY P.Classification;
```

4.2 The Proposed RAG-based Techniques

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.

5 Experiments with an RW-RDB

5.1 Experimental Setup

Benchmark. The experiments used the RW-RDB benchmark defined in Section 3, with the partially extended views, which achieved the best performance in [11].

Performance Indicator. The experiments used the *accuracy* of a given text-to-SQL strategy over the benchmark, 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 *ground truth* SQL queries, entirely based on column and table values, and not just column and table names. Therefore, a text-to-SQL tool may generate SQL queries over the relational schema or any set of views, and the resulting SQL queries may be compared with the ground truth SQL queries based on the results returned. The results of the automated procedure were manually checked to eliminate false positives and false negatives.

Setup configurations. The experiments were based on a text-to-SQL implementation using LangChain SQLQueryChain which automatically extracts metadata from the database, creates a prompt with the metadata and passes it to the LLM. This chain greatly simplifies creating prompts to access databases through views since it passes a view specification as if it were a table specification. This strategy was adopted because it proved inexpensive and had an accuracy compared with much more complex text-to-SQL tools [12].

The multi-attribute synthetic dataset had 46,215 pairs created on top of the partially extended views, whereas the single-attribute synthetic dataset had 24,861 pairs. The usual cosine similarity function was adopted to compare the user's NL question and the NL questions in the dataset.

The experiments tested several configurations, involving different models, strategies, and sample sizes, detailed in the next section to avoid repetition.

5.2 Results

Table 2 shows the results obtained with the different setup configurations. Column **Model** indicates the models used; column **#Samples**, the number of samples retrieved from the synthetic dataset; columns under **#Correct Predicted Queries**, the number of simple, medium, or complex SQL queries correctly synthesized; columns under **Accuracy**, the accuracy results for simple, medium, or complex SQL queries, recalling that the benchmark had 33 simple, 33 medium, and 34 complex NL questions. The lines of Table 2 should be read as follows:

- Lines 1 and 2 show the results when the LLM (GPT-3.5 or GPT-4) is prompted with the relation tables. These are the baselines.
- Lines 3 and 4 show the results using the RAG technique, with the multi-attribute synthetic dataset, and without schema information. The experiments used GPT-3.5-turbo and GPT-4, and retrieved the top 15 most similar pairs from the synthetic dataset.
- Lines 5 and 6 show the results, obtained in [11], using the LLM-friendly partially extended views, without the RAG-based technique.
- Lines 7 and 8 show the results using the RAG-based technique, with the multi-attribute synthetic dataset, combined with the LLM-friendly partially extended views. The experiments used GPT-3.5-turbo-16K and GPT-4-32K, since the prompt turned out to be large, and retrieved the top-8 most similar pairs from the dataset, as well as three rows for each table as samples.

– Lines 9 and 10 show the results using the same configuration as in Lines 7 and 8, except that the RAG-based technique used the single-attribute synthetic dataset.

The rest of this section discusses the results of the setup configurations that use the RAG-based technique in more detail.

Table 2. Results for the different setup configurations – RW-RDB.

#Line	Model (all with SQLQueryChain)	#Samples	#Corrrect Predicted Queries				Accuracy			
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Total
	Relational Schema									
1	GPT-3.5	-	24	8	4	36	0,73	0,24	0,11	0,36
2	GPT-4	-	22	11	8	41	0,67	0,33	0,23	0,41
	RAG-Based Techniques Only									
3	GPT-3.5-turbo	Top 15	24	10	3	37	0,73	0,30	0,09	0,37
4	GPT-4	Top 15	26	7	9	42	0,79	0,21	0,26	0,42
	Partially Extended Conceptual Schema Views Only									
5	GPT-3.5	-	26	18	8	52	0,79	0,54	0,23	0,52
6	GPT-4	-	30	25	19	74	0,91	0,76	0,56	0,74
	RAG-Based Techniques with the Multi-Attribute Synthetic Dataset and the Partially Extended Conceptual Schema Views									
7	GPT-3.5-turbo-16K	Top 8	30	16	9	55	0,91	0,48	0,26	0,55
8	GPT-4-32K	Top 8	31	26	19	76	0,94	0,79	0,56	0,76
	RAG-Based Techniques with the Single-Attribute Synthetic Dataset and the Partially Extended Conceptual Schema Views									
9	GPT-3.5-turbo-16K	Top 8	30	20	11	61	0,91	0,61	0,32	0,61
10	GPT-4-32K	Top 8	32	24	23	79	0,97	0,73	0,68	**0,79**

RAG-based technique only. Consider first the results for the configurations using SQLQueryChain over the partially extended views, with the RAG-based technique using the multi-attribute synthetic dataset, but without passing any schema information (Lines 3 and 4). GPT-3.5 and GPT-4 had relatively low overall accuracy – 37% and 42%, respectively. GPT-4 performed better on simple and complex queries, while GPT-3.5 performed better on medium queries. GPT-4 correctly answered 79% of the simple queries.

In summary, the first experiment suggested that:

– The RAG-based technique provides sufficient information for the LLM to process most of the simple NL questions.
– In more complex NL questions, the examples provided by the RAG-based technique may not include all the necessary information.
– Without knowing the database schema, the LLM may "hallucinate" by using tables and columns that do not exist, or using tables mentioned in the examples but that do not correspond to the user NL question.

Schema information is therefore necessary for generating more complex SQL queries – just the RAG-based technique was insufficient.

RAG-based technique with LLM-friendly partially extended views. Consider now the results for the configurations using the RAG-based technique combined with LLM-friendly partially extended views (Lines 7–10). The best result was obtained with the single-attribute synthetic dataset (lines 9 and 10).

GPT-4 with the top-8 samples had the best performance with a 79% accuracy; it surpassed the previous best configuration using only the LLM-friendly partially extended views (Line 6), which achieved a 74% accuracy. For simple NL questions, this configuration achieved the best performance, with a 97% accuracy. For complex NL questions, it also achieved the best performance with a 68% accuracy; this represents a significant improvement over the previous best approach for complex NL questions, which used only the LLM-friendly partially extended views (Line 6), and achieved a 56% accuracy. However, for medium NL questions, the configuration actually had a lower accuracy than the previous best configuration for medium NL questions (Line 6), as well as lower than RAG with the multi-attribute synthetic dataset (Line 8).

In summary, the second experiment suggested that the RAG-based technique:

- Allowed an LLM to understand the "vocabulary of the database schema". For example, in schema linking, it enabled the LLM to correctly generate SQL queries in ambiguous NL questions such as "out of date" (`out_of_date = yes`) and "canceled orders" (`order.status = 'Canceled'`).
- Increased accuracy to 97% on simple NL questions and to 68% on complex NL questions, whereas the previous best approach reached 91% and 56%, respectively.

Thus, the second experiment suggested that the RAG-based technique with the single-attribute synthetic dataset helped achieve non-trivial improvements on previous results, obtained using only LLM-friendly partially extended views.

6 Experiments with Mondial

Benchmark. The second set of experiments used the Mondial database[7] and 100 NL questions and their *ground truth* translations to SQL queries[8], divided into 34 simple, 33 medium, and 33 complex questions. Mondial stores geographic data and is openly available. It has a total of 47.699 instances; the relational schema has 46 tables, with a total of 184 columns and 49 foreign keys, some of which are multi-column.

Performance Indicator. (Same as in Section 5.1).

Setup configurations. The experiments tested two configurations based on SQLQueryChain, using GPT-3.5-turbo-16K and GPT-4, with RAG on a multi-attribute synthetic dataset with 60,000 pairs, created on top of the Mondial relational schema and associated documentation.

Results. Lines 1 and 2 of Table 3 repeat the best results obtained in [12] for Mondial without using RAG, taken as the baselines; Lines 3 and 4 show the results for the two configurations with RAG. The results corroborate the findings reported in Section 5.2. The RAG-based technique with GPT-4 obtained an accuracy of

[7] https://www.dbis.informatik.uni-goettingen.de/Mondial/
[8] Available on request.

86% (Line 4), surpassing the previous best result of 78% (Line 2), obtained using C3 with GPT-4. The RAG-based technique using GPT-3.5-turbo-16K achieved an accuracy of 72% (Line 3), surpassing all previous strategies tested, except for the RAG-based technique with GPT-4 and C3 with GPT-4. The use of RAG substantially improved the accuracy for medium and complex NL questions. But, for simple NL questions, the multiple similar examples confused the LLM.

Table 3. Results for the different setup configurations – Mondial.

#Line	Strategy / Model	#Samples	#Corrrect Predicted Queries				Accuracy			
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Total
	Relational Schema									
1	SQLQueryChain with GPT-3.5-turbo-16k	3	25	22	13	60	0,76	0,67	0,38	0,60
2	C3 with GPT-4	-	29	25	24	78	0,88	0,76	0,71	0,78
	RAG-Based Techniques with the Multi-Attribute Synthetic Dataset and the Relational Schema									
3	SQLQueryChain with GPT-3.5-turbo-16K	Top 8	28	23	21	72	0,85	0,70	0,62	**0,72**
4	SQLQueryChain with GPT-4	Top 8	28	31	27	86	0,85	0,94	0,79	**0,86**

7 CONCLUSIONS

This paper argued that a RAG-based technique provides a robust strategy to construct database-specific text-to-SQL tools for RW-RDBs. The key step is based on an algorithm that samples the RW-RDB schema and the associated documentation, and calls an LLM to create NL questions and their translation to SQL from the sampled data. By varying how the sampling works, this step generates a dataset containing pairs of NL question/SQL query that give examples of how to create SQL queries with several joins over the RW-RDB adopted, as well as examples that expose the semantics of the data stored in the RW-RDB. The algorithm is generic and applicable to other RW-RDBs.

To argue in favor of the RAG-based technique, the paper included experiments with the same RW-RDB and the same set of questions as in [11], and a text-to-SQL prompt strategy, implemented with LangChain using GPT-3.5 and GPT-4, which includes the RAG-based technique. The results for the RW-RDB suggested that the RAG-based technique improved accuracy, especially for complex NL questions. The results for Mondial corroborated these findings.

In future work, we plan to expand the experiment to other RW-RDBs, and with variations of the RAG-based technique, including changing the similarity function and working with a different strategy that retrieves a diversified list of pairs, rather than just the top-n. We also plan to expand the approach to cover different user groups, with distinct and often contradictory vocabularies.

ACKNOWLEDGEMENTS

This work was partly funded by FAPERJ under grant E-26/202.818/2017; by CAPES under grants 88881.310592-2018/01, 88881.134081/2016-01, and 88882.164913/2010-01; by CNPq under grant 302303/2017-0; and by Petrobras.

References

1. 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
2. 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
3. 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 (01 2021). https://doi.org/10.18653/v1/2021.emnlp-main.702
4. Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Guo, Q., Wang, M., Wang, H.: Retrieval-augmented generation for large language models: A survey. arXiv preprint (2024). https://doi.org/10.48550/arXiv.2312.10997
5. 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. arXiv preprint (2023). https://doi.org/10.48550/arXiv.2307.05074
6. Katsogiannis-Meimarakis, G., Koutrika, G.: A survey on deep learning approaches for text-to-sql. The VLDB Journal **32**(4), 905–936 (2023). https://doi.org/10.1007/s00778-022-00776-8
7. Kim, H., So, B.H., Han, W.S., Lee, H.: Natural language to sql: Where are we today? Proc. VLDB Endow. **13**(10), 1737–1750 (jun 2020). https://doi.org/10.14778/3401960.3401970
8. 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
9. Li, J., et al.: Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. arXiv preprint (2023). https://doi.org/10.48550/arXiv.2305.03111
10. Manning, C.D.: Human Language Understanding & Reasoning. Daedalus **151**(2), 127–138 (05 2022). https://doi.org/10.1162/daed_a_01905
11. 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. pp. 800–806 (2024)
12. 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. pp. 61–72 (2024)
13. 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)
14. 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