

## LLM-Based Text-to-SQL for Real-World Databases

This Accepted Manuscript (AM) is a PDF file of the manuscript accepted for publication after peer review, when applicable, but does not reflect post-acceptance improvements, or any corrections. Use of this AM is subject to the publisher's embargo period and AM terms of use. Under no circumstances may this AM be shared or distributed under a Creative Commons or other form of open access license, nor may it be reformatted or enhanced, whether by the Author or third parties. By using this AM (for example, by accessing or downloading) you agree to abide by Springer Nature's terms of use for AM versions of subscription articles: <https://www.springernature.com/gp/open-research/policies/accepted-manuscript-terms>

The Version of Record (VOR) of this article, as published and maintained by the publisher, is available online at: <https://doi.org/10.1007/s42979-025-03662-6>. The VOR is the version of the article after copy-editing and typesetting, and connected to open research data, open protocols, and open code where available. Any supplementary information can be found on the journal website, connected to the VOR.

For research integrity purposes it is best practice to cite the published Version of Record (VOR), where available (for example, see ICMJE's guidelines on overlapping publications). Where users do not have access to the VOR, any citation must clearly indicate that the reference is to an Accepted Manuscript (AM) version.

# LLM-based Text-to-SQL for Real-World Databases

Eduardo R. Nascimento<sup>1</sup>, Grettel García<sup>1</sup>, Yenier T. Izquierdo<sup>1</sup>,  
 Lucas Feijó<sup>1</sup>, Gustavo M.C. Coelho<sup>1</sup>, Aiko R. de Oliveira<sup>3</sup>,  
 Melissa Lemos<sup>1,3</sup>, Robinson L.S. Garcia<sup>2</sup>, Luiz A. P. Paes Leme<sup>4</sup>,  
 Marco A. Casanova<sup>1,3\*</sup>

<sup>1</sup>Instituto Tecgraf, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil.

<sup>2</sup>Petrobras, Rio de Janeiro, 20031-912, RJ, Brazil.

<sup>3</sup>Depto. Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil.

<sup>4</sup>Instituto de Computação, UFF, Niterói, 24210-310, RJ, Brazil.

\*Corresponding author(s). E-mail(s): [casanova@inf.puc-rio.br](mailto:casanova@inf.puc-rio.br);

Contributing authors: [rogerrsn@tecgraf.puc-rio.br](mailto:rogerrsn@tecgraf.puc-rio.br);

[ggarcia@tecgraf.puc-rio.br](mailto:ggarcia@tecgraf.puc-rio.br); [ytorres@tecgraf.puc-rio.br](mailto:ytorres@tecgraf.puc-rio.br);

[lucasfeijo@tecgraf.puc-rio.br](mailto:lucasfeijo@tecgraf.puc-rio.br); [gustavocoelho@tecgraf.puc-rio.br](mailto:gustavocoelho@tecgraf.puc-rio.br);

[aramalho@inf.puc-rio.br](mailto:aramalho@inf.puc-rio.br); [melissa@tecgraf.puc-rio.br](mailto:melissa@tecgraf.puc-rio.br);

[robinson.garcia@petrobras.com.br](mailto:robinson.garcia@petrobras.com.br); [lapaesleme@ic.uff.br](mailto:lapaesleme@ic.uff.br);

## Abstract

Text-to-SQL refers to the task defined as “*given a relational database  $D$  and a natural language sentence  $S$  that describes a question on  $D$ , generate an SQL query  $Q$  over  $D$  that expresses  $S$* ”. Several LLM-based text-to-SQL tools, that is, text-to-SQL tools that explore Large Language Models (LLMs), emerged that outperformed previous approaches on well-known benchmarks. This article first shows that the performance of a selected set of LLM-based text-to-SQL tools is, however, significantly less when run on two challenging databases with a large number of tables, columns, and foreign keys. A closer analysis reveals that one of the problems lie in that the relational schema is an inappropriate specification of the database from the point of view of the LLM. The article then introduces database specifications 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 database values. The article proceeds to show that the use of a set of LLM-friendly views and data samples considerably improves the performance of a text-to-SQL prompt strategy over a real-world database. This result suggests that real-world databases require rethinking how

schema specifications should be passed to the LLM to recover state-of-the-art performance.

**Keywords:** Text-to-SQL, GPT, Large Language Models, Natural Language, Relational Databases.

**CCS:** I.2 , I.2.5 , I.2.7 , H.2 , H.2.8 , H.3 , H.3.3

## 1 Introduction

Text-to-SQL [1], also referred to as NL2SQL [2], is the task defined as “*given a relational database  $D$  and a natural language (NL) sentence  $S$  that describes a question on  $D$ , generate an SQL query  $Q$  over  $D$  that expresses  $S$* ”. Numerous tools have addressed this task with relative success [1–3] over well-known benchmarks, such as Spider – Yale Semantic Parsing and Text-to-SQL Challenge [4] and BIRD – BIG Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation [5]. Recently, several LLM-based text-to-SQL tools, that is, text-to-SQL tools that explore Large Language Models (LLMs), emerged that outperformed previous approaches on both benchmarks. However, when run over industrial-size, real-world databases, the performance of some LLM-based text-to-SQL tools is significantly less than that reported in the Spider and BIRD leaderboards.

Indeed, real-world databases are challenging for at least four reasons:

- C1. 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.
- C2. 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.
- C3. The data semantics are 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.
- C4. Metadata and data are often ambiguous, which influences the behavior of an LLM-based text-to-SQL tool, leading to unexpected results.

To argue that statement C1 is indeed a challenge, the article assesses the performance of several LLM-based text-to-SQL strategies over relational databases with large schemas, with several hundred objects, counting tables, columns, and foreign keys.

In more detail, the article adopts two challenging databases: a proprietary industrial database, IndDB (the actual name has been anonymized), and an openly available database, Mondial. IndDB is an oil and gas industrial asset integrity management database, in production at an industrial company. It features a relational schema with 27 tables, 585 columns, and 30 foreign keys (some of which are multi-column); the

largest table has 81 columns. Thus, IndDb has nearly 640 hundred objects. IndDB features a database keyword search tool, DANKE [6], which compiles SQL queries from keyword queries, enhanced with attribute filtering and limited forms of aggregation. However, since IndDB is proprietary, the experiments cannot be replicated. Thus, the article also considers Mondial, an openly-available database storing geographic data, with a total of 47.699 instances; the relational schema has 46 tables, with a total of 184 columns and 49 foreign keys (again, some of which are multi-column) – a total of nearly 280 objects. Mondial was chosen precisely because it has a large schema, which is challenging for text-to-SQL, and is openly available, which permits replicating the results. A version of Mondial with 34 tables is also part of the BIRD benchmark. The *IndDB* and the *Mondial benchmarks* comprise these databases, each accompanied by a suite of 100 NL questions, closely related to those observed in practice, and their SQL ground truth translations.

Table 1 lists the text-to-SQL strategies assessed. They include the database keyword query tool DANKE, developed to access the industrial database IndDB, a strategy that combines DANKE with keyword extraction from an NL question [7], SQLCoder<sup>1</sup> based on an LLM fine-tuned to text-to-SQL, “LangChain SQLQueryChain”<sup>2</sup> with and without samples, “C3 + ChatGPT + Zero-Shot” [8], “DIN-SQL + GPT-4” [9], and “C3+DIN”, a new strategy that combines components of C3 and DIN-SQL. The article first analyses these seven text-to-SQL strategies over the Mondial benchmark, selects the three best-performing, and tests them over the IndDB benchmark. Except for DANKE and SQLCoder, the article tested each strategy with GPT-3.5 Turbo and GPT-4.

The experiments show that, for the Mondial benchmark: the top-5 strategies with respect to overall accuracy used GPT-4; C3 had the best overall accuracy of 0.78; and SQLCoder had a limited overall accuracy of 0.35. Using the IndDb benchmark, the overall accuracy of the strategies tested was very low: only LangChain SQL-QueryChain using samples achieved an overall accuracy above 30% – 36% for GPT-3.5 and 41% with GPT-4, which are still low. These results corroborate that statement C1 is indeed a challenge. In general, the limited performance of text-to-SQL strategies, especially for IndDB, calls for different text-to-SQL approaches.

The article proceeds to address Challenges C1, C2, and C3 (Challenge C4 is outside the scope of this article). It argues that the text-to-SQL task can be greatly facilitated by a database specification that provides:

- *LLM-friendly views* that map (fragments of) the database schema to terms close to the terms users frequently adopt and that try to predefine frequently used joins.
- *LLM-friendly descriptions* of the data values.

LLM-friendly views are nothing but the familiar concept of views, designed to present (fragments of) the relational schema (that is, database metadata) to the LLM. As such, they can be implemented with the usual DBMS mechanisms, within the database. In general, LLM-friendly descriptions refer to constructs that capture the

<sup>1</sup><https://huggingface.co/defog/sqlcoder-34b-alpha>

<sup>2</sup>[https://python.langchain.com/v0.1/docs/use\\_cases/sql/prompting/](https://python.langchain.com/v0.1/docs/use_cases/sql/prompting/)

**Table 1** Summary of the strategies tested [10].

1) <b>“DANKE”</b> – The database keyword search tool is the current interface for IndDB.
2) <b>“Manual prompt chain + DANKE”</b> – The LLM extracts keywords from the NL question, and passes the keywords to KwS.Tool; tested with GPT-3.5-turbo and GPT-4.
3) <b>“SQLCoder”</b> – Defog’s tool for text-to-SQL, using a special-purpose LLM.
4) <b>“LangChain SQLQueryChain”</b> – The LangChain SQLQueryChain processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
5) <b>“DIN-SQL”</b> – An implementation of “DIN-SQL” that allows the use of different LLMs; tested with GPT-3.5-turbo-16k and GPT-4.
6) <b>“C3”</b> – An implementation of “C3” that allows the use of different LLMs; tested with GPT-3.5-turbo and GPT-4.
7) <b>“C3+DIN”</b> – The C3+DIN tool introduced in Section 4.4; tested with GPT-3.5-turbo-16k and GPT-4.

data semantics. In this article, row samples of the tables involved in an NL question will be used to convey the data semantics to the LLM.

To evaluate the approach proposed to address Challenges C1, C2, and C3, the article expands on the IndDB benchmark with three sets of LLM-friendly views. The article then investigates the performance of LangChain SQLQueryChain using GPT-3.5 and GPT-4 over this extended benchmark. The prompt strategy now includes row samples to help the LLM capture the data semantics. The results suggest that the use of a set of LLM-friendly views and data samples is sufficient for LangChain SQLQueryChain to achieve an accuracy that is significantly better than the performance obtained using the original IndDB relational schema.

The use of views for the Mondial database had little impact, since the Mondial relational schema uses table and column names which are familiar to end-users. Thus, these results are not included in this article.

This article is an extended version of [10] and incorporates the contributions first reported in [11]. It describes a revised set of experiments, based on a curated set of NL questions for IndDB and including the notions of LLM-friendly views and LLM-friendly descriptions.

The article is organized as follows. Section 2 covers related work. Section 3 describes the IndDB and the Mondial benchmarks. Section 4 summarizes the strategies tested. Section 5 details the experiments to evaluate the effect of schema complexity. Section 6 describes the experiments to evaluate the effect of using views. Section 7 contains the conclusions.

## 2 Related Work

### 2.1 Text-to-SQL Datasets

The Spider – Yale Semantic Parsing and Text-to-SQL Challenge [4] 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<sup>3</sup>; and 90 databases based on WikiSQL, with about 500 tables in about 90 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 of the prediction and the ground truth SQL queries match exactly; *exact matching* measures whether the predicted SQL query as a whole is equivalent to the ground truth SQL 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. In the number of tables, the largest five are: `baseball_1`, with 25 tables, `cre_Drama_Workshop_Groups`, with 18 tables, and `cre_Theme_park`, `imdb`, and `sakila_1`, with 16 tables. 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.

Released in July 2024, Spider2-V<sup>4</sup> is a multimodal agent benchmark spanning the entire data science and engineering workflow and does not replace Spider as a text-to-SQL benchmark.

BIRD – BIG Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation [5] 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 tries to bridge the gap between text-to-SQL research and real-world applications by exploring three additional challenges: dealing with large and messy database values, external knowledge inference, and optimizing SQL execution efficiency. However, as mentioned in the introduction, BIRD still has only two databases out of 73 in the training dataset, with more than 30 tables.

WikiSQL [12] 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.

Chase [13] is a dataset for the cross-database context-dependent Text-to-SQL problem in Chinese. It consists of 5,459 question sequences (17,940 questions) over 280 databases, in which only 35% of questions are context-independent, and 28% of SQL queries are easy. Each question in Chase has rich semantic annotations, including its SQL query, contextual dependency, and schema linking.

<sup>3</sup><http://www.databaseanswers.org/>

<sup>4</sup><https://spider2-v.github.io>

SQL-Eval [14] is a framework<sup>5</sup> that evaluates the correctness of text-to-SQL strategies, created during the development of the SQLCoder.

Finally, the `sql-create-context`<sup>6</sup> 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.

## 2.2 Text-to-SQL Tools

LangChain<sup>7</sup> is a generic framework that offers several predefined strategies to build and run SQL queries based on NL prompts. Section 4.3 reviews in more detail LangChain.

The Spider Web site<sup>8</sup> publishes a leaderboard with the best-performing text-to-SQL tools. At the time of this writing, in terms of Execution with Values, the top 6 tools 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 [15]; “DPG-SQL + GPT-4 + Self-Correction” (no reference available at the time of writing); “DIN-SQL + GPT-4” [9]; “Hindsight Chain of Thought with GPT-4” (no reference available at the time of writing); and “C3 + ChatGPT + Zero-Shot” [8].

The BIRD Web site<sup>9</sup> also publishes a leaderboard with the best-performing tools. At the time of this writing, in terms of Execution Accuracy, the top 5 tools were: “SFT CodeS-15B” and “SFT CodeS-7B” (no reference available at the time of writing); “DAIL-SQL + GPT-4”; “DIN-SQL + GPT-4”; and GPT-4.

Section 4 presents the details of “DIN-SQL + GPT-4” and “C3 + ChatGPT + Zero-Shot”. These tools were selected since their code was available when the experiments reported in this article were designed.

Other text-to-SQL tools include the Defog SQLCoder, outlined in Section 4.2. The “60 Top AI Text To SQL Bot Tools” Web site<sup>10</sup> lists other AI tools for text-to-SQL.

Finally, the DB-GPT-Hub<sup>11</sup> is a project exploring how to use LLMs for text-to-SQL. The project contains data collection, data preprocessing, model selection and building, and fine-tuning of weights, including LLaMA-2, and the evaluation of several LLMs fine-tuned for text-to-SQL.

## 3 Benchmark Adopted

A *benchmark* for the text-to-SQL task is a pair  $B = (D, \{(L_i, G_i)/i = 1, \dots, n\})$ , where  $D$  is a database and, for  $i = 1, \dots, n$ ,  $L_i$  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 article, a benchmark is meant exclusively for testing text-to-SQL tools; it is not designed to train such tools.

<sup>5</sup> Available at <https://github.com/defog-ai/sql-eval>

<sup>6</sup> <https://huggingface.co/datasets/b-mc2/sql-create-context>

<sup>7</sup> <https://python.langchain.com>

<sup>8</sup> <https://yale-lily.github.io/spider>

<sup>9</sup> <https://bird-bench.github.io>

<sup>10</sup> <https://topai.tools/s/Text-to-SQL-bot>

<sup>11</sup> <https://github.com/eosphoros-ai/DB-GPT-Hub>

### 3.1 Primary Benchmarks

The primary benchmarks adopted in this article are based on the proprietary industrial database, IndDB, and on the Mondial database, each with a set of 100 NL questions and their translations to SQL.

IndDB is an oil and gas industrial asset integrity management database in production at an industrial company. It features a relational schema with 27 tables, 585 columns, and 30 foreign keys (some multi-column); the largest table has 81 columns.

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.

However, since IndDB is proprietary and is not openly available, the experiments cannot be replicated. Thus, the article also considers a version of Mondial<sup>12</sup>, with a set of 100 NL questions and their translations to SQL, 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<sup>13</sup> has 46 tables, with a total of 184 columns and 49 foreign keys, some of which are multi-column.

Table 2 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. Note that, on average, the medium and complex queries for IndDB have roughly twice the number of joins and filters as the medium and complex queries for Mondial.

**Table 2** Basic statistics of the sets of queries [10].

		IndDB				Mondial			
	Query Type	#cols.	#joins	#filters	#aggr	#cols.	#joins	#filters	#aggr
Average	complex	2,00	3,03	2,55	0,39	1,09	1,41	1,65	0,38
	medium	1,44	2,47	1,68	0,29	1,18	0,79	0,85	0,30
	simple	1,30	0,21	0,82	0,15	1,33	0,42	0,73	0,12
Maximum	complex	19	5	4	2	2	4	5	1
	medium	9	5	3	2	2	3	1	2
	simple	6	2	2	1	4	3	2	1

### 3.2 The IndDB Benchmark With Views

To verify how the LLM-friendly views and data samples affect the text-to-SQL task, the IndDB benchmark is augmented with three sets of LLM-friendly views of increasing complexity:

<sup>12</sup><https://www.dbis.informatik.uni-goettingen.de/Mondial/>

<sup>13</sup>The Mondial referential dependencies diagram can be found at <https://www.dbis.informatik.uni-goettingen.de/Mondial/mondial-abh.pdf>

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

The ground truth SQL queries,  $G = \{G_1, \dots, G_{100}\}$ , were manually redefined over the conceptual schema views so that the execution of  $G_i$  returns the expected answer to the NL question  $L_i$ .

The NL questions classification is now 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).

## 4 Strategies Tested

This section outlines the strategies tested, which are grouped into four classes (see Table 1): DANKE-based strategies, SQLCoder, LangChain SQLQueryChain, and C3 and DIN-based strategies.

### 4.1 DANKE-based Strategies

DANKE [6] is an automatic, schema-based tool that supports keyword query processing for both the relational and RDF environments [16]. DANKE is in production at an oil company as an interface for several databases, including IndDB.

DANKE supports expressions that contain keywords along with filters and certain forms of aggregation, embedded along the keywords, and some syntactical sugar to enhance readability. An example of an expression that DANKE accepts is “What is the total number of cities in Botswana?”, which corresponds to the keyword query “total cities Botswana”. Note that DANKE recognizes and processes the aggregation “total cities” and the restriction “Country\_name=Botswana”, and ignores the phrase “What is”, the article “the”, and the preposition “of”, treated as syntactical sugar.

The core engine of DANKE first constructs a Steiner tree that covers a set of nodes (relation schemes or RDF classes) whose instances match the largest set of keywords [17]. It then compiles the keyword-based query into an SQL (or SPARQL) query that includes conditions that represent keyword matches and joins. Without such joins, an answer would be a disconnected set of tuples (or nodes of the RDF graph), which hardly makes sense.

The “DANKE” strategy listed in Table 1 passes the NL question directly to DANKE.

The “Manual prompt chain + DANKE” strategy [7] uses an LLM to extract keywords from an NL question through a sequence of 4 steps, each with a specific prompt: (1) identify the relevant entities; (2) identify the relevant properties; (3) generate

keyword query candidates; and (4) select the best query  $Q$ . Then,  $Q$  is passed to DANKE, which processes  $Q$  as explained above. The experiments tested this strategy with GPT-3.5-turbo and GPT-4.

## 4.2 SQLCoder

SQLCoder<sup>14</sup> is a specialized text-to-SQL model, open-sourced under the Apache-2 license. The sqlcoder-34b-alpha model features 34B parameters and was fine-tuned on a base CodeLlama model, on more than 20,000 human-curated questions, classified as in Spider, based on ten different schemas. The term SQL Coder will refer to the model and the tool based on the model.

The training dataset 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 SQLCoder.

The accuracy of SQLCoder on the Defog’s dataset is promising: defog-sqlcoder-34b achieved 84.0%, whereas gpt4-turbo-2023-11-09 achieved 82.5%.

## 4.3 LangChain SQLQueryChain

LangChain<sup>15</sup> 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.

The experiments tested LangChain SQLQueryChain with GPT-3.5-turbo-16k and GPT-4.

## 4.4 C3 and DIN-based Strategies

### 4.4.1 C3

“C3 + ChatGPT + Zero-Shot” [8] (or briefly C3) is a prompt-based strategy, originally defined for ChatGPT, that uses only approximately 1,000 tokens per query and achieves a better performance than fine-tuning-based methods. C3 has three key components: *Clear Prompting* (CP); *Calibration with Hints* (CH); *Consistent Output* (CO).

---

<sup>14</sup><https://huggingface.co/defog/sqlcoder-34b-alpha>

<sup>15</sup><https://docs.langchain.com>

*Clear Prompting* addresses two problems: (1) the size of the database schema may exceed the prompt limit; (2) a prompt with too many tables and columns may confuse ChatGPT. Clear prompting then recalls relevant tables and columns (to the NL question) and adds them to the prompt, along with the schema.

*Calibration with Hints* avoids errors caused by certain biases inherent in ChatGPT: to select columns that are relevant to the question but not required; to use LEFT JOIN, OR, and IN incorrectly. Calibration with Hints then instructs ChatGPT to follow two “debias” hints: (1) select only the necessary column; (2) avoid misusing SQL constructs.

*Consistent Output* tries to avoid the problem that the output of ChatGPT is unstable due to the inherent randomness of LLMs. The proposed solution samples multiple reasoning paths to generate several SQL queries, executes the SQL queries on the database and collects the execution outcomes, and uses a voting mechanism on the results to identify the most consistent SQL.

At the time of writing, C3 was the sixth strategy listed in the Spider leaderboard, achieving 82.3% in terms of execution accuracy on the test set. It outperformed state-of-the-art fine-tuning-based approaches in execution accuracy on the test set, while using only approximately 1,000 tokens per query.

The experiments tested C3 with GPT-3.5-turbo and GPT-4.

#### 4.4.2 DIN

“DIN-SQL + GPT-4” [9] (or briefly DIN) uses only prompting techniques and decomposes the text-to-SQL task into four steps: *schema linking*; *query classification and decomposition*; *SQL generation*; and *self-correction*.

*Schema Linking* includes ten randomly selected samples from the training set of the Spider dataset and follows the chain-of-thought template. The prompt begins with “Let’s think step by step,”...; for the column names mentioned in the question, the corresponding columns and their tables are selected from the schema; possible entities and cell values are also extracted from the question.

*Classification and Decomposition* classifies each query into: *easy* – single-table queries that can be answered without joins or nesting; *non-nested* – queries that require joins but no sub-queries; *nested* – queries that require joins, sub-queries, and set operations.

*SQL Generation* depends on the query classification. A simple few-shot prompting with no intermediate steps is adequate for easy queries. For non-nested complex queries, it uses NatSQL as an intermediate representation, removes operators JOIN ON, FROM, GROUP BY, and set operators, and merges the HAVING and WHERE clauses. Briefly, for nested complex queries, it breaks down the problem into multiple steps; the prompt for this class is designed in a way that the LLM should first solve the sub-queries and then use them to generate the final answer.

Finally, *Self-Correction* addresses the problem that the generated SQL queries can sometimes have missed or redundant keywords such as DESC, DISTINCT, and aggregation functions. To solve this problem, the self-correction step instructs the LLM to correct those minor mistakes by a zero-shot setting, where only the buggy code is passed to the LLM, which is asked to fix the bugs.

When released, “DIN-SQL + GPT-4” was the top-performing tool listed in the Spider Leaderboard, achieving 85.3% in terms of execution accuracy.

The experiments tested DIN with GPT-3.5-turbo-16k and GPT-4.

#### 4.4.3 C3+DIN

“C3+DIN” combines some of the C3 and DIN strategies and decomposes the text-to-SQL task into six steps: *database schema description*; *clear prompting and schema linking*; *classification and decomposition*; *calibration with hints*; *SQL generation*; and *self-correction*.

*Database schema description* uses LangChain’s SQLDatabase module to access database metadata. This metadata is manipulated to represent the schema in an automated way.

*Clear prompting and schema linking* takes advantage of DIN’s chain of thought for schema linking, but uses C3’s clear prompting to pass only the tables and columns relevant to the query. Hence, “C3 + DIN” uses fewer tokens to represent the schema in the prompt.

*Classification and decomposition* uses the DIN’s classification strategy, which is important for SQL generation, since it uses different prompts for each class. Additionally, the prompts help detect tables that should be joined and nested queries by detecting subqueries contained in the main query.

*Calibration with hints* uses C3’s calibration, which incorporates prior knowledge of the LLM. Before the SQL generation step, “C3 + DIN” provides hints to help the model generate SQL queries that align more closely with the desired output.

*SQL generation* is based on DIN’s chain of thought [18], where some thought-chain demonstrations are provided as stimulus examples. This significantly improves the ability of LLMs to perform complex reasoning. Furthermore, DIN’s SQL Generation applies different few-shot prompts for each query class. Thus, “C3 + DIN” adopts this approach to generate SQL code.

*Self-correction* incorporates DIN’s self-correction mechanism, which seeks to correct the SQL code in a simple way by passing some hints to the model along with the database schema. Indeed, LLMs have hallucination problems, that is, they can generate text that does not make sense [19].

The experiments tested “C3+DIN” with GPT-3.5-turbo-16k and GPT-4.

## 5 Experiments to Evaluate the Effect of Schema Complexity

### 5.1 Configurations

The experiments tested the strategies summarized in Table 1 for the 100 NL questions and their translations to SQL over the Mondial and IndDB databases, stored in Oracle. The foreign keys of Mondial were used, but not those of IndDB, which were not available for the experiments.

Except for DANKE and 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.

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.

## 5.2 Evaluation Procedure

Let  $B = (D, \{(L_i, G_i)/i = 1, \dots, n\})$  be a benchmark. Recall that  $L_i$  is an NL question and  $G_i$  is the corresponding ground truth SQL query. Let  $P_i$  be the SQL query *predicted* by a text-to-SQL strategy for  $L_i$ . 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 the same columns, nor do they have 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  $APT_i$  iff  $GC_k$ , 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 coefficient (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 four or fewer columns (indeed,  $4 * 0.80 = 3.20$  is rounded up to 4, that is,  $GT_i$  must have all four columns matching some column of  $PT_i$ , and likewise for a smaller number of columns).

Now, from column “#cols” of Table 2, observe that all queries for Mondial have a result with at most four columns. Hence, setting  $tn = 0.80$  implies that all columns of  $GT_i$  must match a column of  $PT_i$ . However, the same is not valid for IndDB. Indeed, from column “#cols” of Table 2, observe that some queries for IndDB have a result

with up to 19 columns. Hence, for such queries, some columns of  $GT_i$  may not match any 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 coefficient (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 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.

### 5.3 Results

Tables 3 and 5 show the results for the Mondial and the IndDB benchmarks, respectively. 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 translate the 100 NL questions, which naturally depends on the HW and SW setup, and should be used only to compare the strategies.

#### 5.3.1 Results for the Mondial Benchmark

**Accuracy.** The top-5 strategies concerning 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. SQLCoder had a limited overall accuracy of 0.35.

As for the query types, C3 with GPT-4 had the best accuracy for complex queries, 0.71; SQLQueryChain with samples had the best accuracy for medium queries, 0.85; and C3+DIN with GPT-4 had the best accuracy for simple queries, 0.91.

**Strategy details.** Experiments with Langchain were divided into two groups: (1) passing the NL question and the schema; and (2) passing the NL question, the schema, and two sample rows from each table. The second group resulted in larger prompts, requiring GPT-3.5-turbo-16k in some cases, while GPT-4 handled large prompts seamlessly.

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, at the time of writing, the output token price of GPT-4 (\$0.06/1K sampled tokens)

**Table 3** Results for the Mondial benchmark [10].

#Line	Model	Accuracy				Input Tkn	Output Tkn	Tokens	Est. Cost	Exec Time	Comm.
		Simple	Medium	Complex	Total						
1	<b>DANKE</b>	0,33	0,09	0,03	0,15	---	---	---	---		
<b>Manual prompt chain + DANKE</b>											
2	GPT-3.5-turbo	0,18	0,30	0,09	0,19	781.151	28.297	809.448	\$1,23		
3	GPT-4	0,33	0,27	0,18	0,26	792.266	36.250	828.516	\$25,94		
4	<b>SQLCoder</b>	0,45	0,39	0,21	0,35	---	---	---	---		
<b>SQLQueryChain (LangChain)</b>											
5	GPT-3.5-turbo	0,76	0,58	0,32	0,55	569.713	2.563	572.276	\$1,72	00:02:15	
6	GPT-4	0,82	0,79	0,47	0,69	569.713	2.643	572.356	\$17,25	00:05:28	
<b>SQLQueryChain (LangChain) - with samples</b>											
7	GPT-3.5-turbo-16k	0,76	0,67	0,38	0,60	782.813	2.501	785.314	\$2,36	00:06:47	(1)(2)(3)
8	GPT-4	0,82	<b>0,85</b>	0,44	0,70	782.624	2.527	785.151	\$23,63	00:09:20	(2)
<b>DIN</b>											
9	GPT-3.5-turbo-16k	0,82	0,45	0,41	0,56	1.431.645	33.050	1.464.695	\$4,43	00:14:54	(1)(4)
10	GPT-4	0,85	0,76	0,50	0,70	1.428.284	32.473	1.460.757	\$44,80	00:45:26	
<b>C3</b>											
11	GPT-3.5-turbo	0,79	0,48	0,38	0,55	175.298	754.619	929.917	\$1,77	01:09:27	(5)
12	GPT-4	0,88	0,76	<b>0,71</b>	<b>0,78</b>	153.705	426.937	580.642	\$30,23	01:20:06	
<b>C3+DIN</b>											
13	GPT-3.5-turbo-16k	0,82	0,52	0,44	0,59	1.147.061	719.891	1.866.952	\$6,32	01:16:52	(1)(5)(6)
14	GPT-4	<b>0,91</b>	0,70	0,50	0,70	1.137.152	401.191	1.538.343	\$58,19	01:48:36	(5)

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) The DIN prompt requires a larger token limit.
- (5) Table and column metadata recall took a long time.

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 shows the error analysis of C3 with GPT-4. When compared with that of C3 for the Spider benchmark [8], 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** Error analysis of the C3 with GPT-4 experiment using the Mondial database [10].

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

The benchmark and experiments using the Mondial database are openly available in the GitHub repository: [https://github.com/dudursn/text\\_to\\_sql\\_chatgpt\\_real\\_world](https://github.com/dudursn/text_to_sql_chatgpt_real_world).

**Table 5** Results for the IndDB benchmark [10].

#Line	Strategy / Model	#Samples	Accuracy				Cost			
			Simple	Medium	Complex	Total	Input Tokens	Output Tokens	Estimated cost	Time
<i>LangChain SQLQueryChain - with samples</i>										
1	GPT-3.5-turbo-16k	(1)	<b>0,73</b>	0,24	0,11	0,36	1011903	5377	\$3,06	00:02:25
2	GPT-4	(1)	0,67	<b>0,33</b>	0,24	<b>0,41</b>	1011903	11405	\$31,04	00:23:05
<i>DIN</i>										
3	GPT-3.5-turbo-16k		0,67	0,06	0,03	0,25	447986	39858	\$4,50	00:17:34
4	GPT-4		0,64	0,09	0,03	0,25	<b>1442688</b>	39509	<b>\$45,65</b>	00:14:53
<i>C3</i>										
5	GPT-3.5-turbo		0,64	0,09	0,06	0,26	291355	<b>498849</b>	\$2,87	02:21:31
6	GPT-4		0,73	0,12	0,03	0,29	292067	266826	\$24,77	<b>02:39:03</b>

(1) The strategies prompted the LLM with 3 instances per table.

### 5.3.2 Results for the IndDB Benchmark

The experiments with the IndDB benchmark tested “LangChain SQLQueryChain” with samples, DIN, and C3, which achieved very good overall accuracy over the Mondial benchmark.

**Accuracy.** The overall accuracy of all strategies tested was very low. Only LangChain SQLQueryChain using samples achieved an overall accuracy above 30% – 36% for GPT-3.5 and 41% with GPT-4, which are still low results.

**Strategy details.** DIN with GPT-4 generated the largest number of input tokens since the IndDB schema has fewer tables than Mondial, but the tables have more columns. C3 generated the largest number of output tokens with GPT-4.

**Analysis of the predicted SQL queries.** The low accuracy results call for a careful comparison of the predicted and the ground truth SQL queries. Rather than going through each of the six configurations (the number of lines of Table 5), the analysis concentrated on LangChain SQLQueryChain with GPT-4 using samples, the best accuracy of the six configurations.

The analysis revealed some recurrent problems with the experiments with IndDB, especially the following three:

1. Many NL questions led to SQL queries with up to 5 joins to gather all the required data (see Table 2).
2. The relational schema had table and column names that did not match terms the user typically adopted for the concepts (and therefore were used to formulate the NL questions).
3. Certain ground truth SQL queries were very specific interpretations of terms in the NL question.

Note that these problems are just manifestations of Challenges C1, C2 and C3 alluded to in Section 1. Indeed, an overall accuracy of 0.41 for LangChain SQLQueryChain using samples and GPT-4 over IndDB is much lower than the overall accuracy of 0.70 that this strategy obtained over Mondial, reported in Table 3. Since the Mondial relational schema uses familiar table and column names, this observation suggests that problem (2) may be blamed, in part, for the lower accuracy of IndDB.

As mentioned in Section 1, the article proceeds to verify that the text-to-SQL task can indeed be facilitated by a database specification that provides: *LLM-friendly views*, that map (fragments of) the database schema to terms close to the terms users frequently adopt and that try to predefine frequently used joins; *LLM-friendly descriptions* of the database values.

## 6 Experiments to Evaluate the Effect of Using Views on IndDB

The experiments tested LangChain SQLQueryChain with GPT-3.5-turbo-16k and GPT-4 and samples against the 100 questions for IndDB, separately for each of the three sets of views outlined in Section 3.2. SQLQueryChain was adopted because it achieved the best overall accuracy in the experiments reported in Section 5.3.2. As mentioned in the introduction, the use of views for Mondial had little impact, since the Mondial relational schema uses table and column names which are familiar to end-users. Thus, these results were not included in this article.

### 6.1 Experimental Setup

Figure 1 illustrates the prompt implemented: (A) contains instructions for the LLM; (B) defines the output format; (C) partly illustrates how the `maintenance_order` view is passed to the LLM as a `CREATE TABLE` statement; (D) shows 3 data samples from the `maintenance_order` view; and (E) passes the NL question.

The evaluation procedure remains the same as in Section 5.2.

The figure shows a prompt template for an LLM. It is divided into five sections, each in a separate box:

- (A)** Instructions for the LLM: "You are an Oracle SQL expert. Given an input question, first create a syntactically correct Oracle SQL query to run, then look at the results of the query and return the answer to the input question. Unless the user specifies in the question a specific number of examples to obtain, don't query for at 0 most results or any using the FETCH FIRST n ROWS ONLY clause as per Oracle SQL. You can order the results to return the most informative data in the database. Never query for all columns from a table. You must query only the columns that are needed to answer the question. Pay attention to use only the column names you can see in the tables below. Be careful to not query for columns that do not exist. Also, pay attention to which column is in which table. Pay attention to use TRUNC(SYSDATE) function to get the current date, if the question involves "today". Generate only the sql query. Don't give the answer and don't explain."
  - Some hints:
    - Don't use double quotes in column name
  - Example:
    - 'SELECT "column\_name" FROM table' should be 'SELECT column\_name FROM table'
    - Don't use LEFT JOIN, only JOIN
- (B)** Use the following format:
 

```
Question: Question here
SELECT
```
- (C)** Only use the following tables:
 

```
CREATE TABLE maintenance_order (
  description VARCHAR(40 CHAR),
  code VARCHAR(30 CHAR),
  status VARCHAR(7 CHAR),
  .
  .
  .
)
```
- (D)** 3 rows from the maintenance\_order table:
 

description	code	status
FT-UC-123101B-04	818190	Active
SYSTEM-5111.03	301063	Active
SYSTEM-5412.03	301063	Active
- (E)** Question: {input}

Fig. 1 Example of a prompt [11].

## 6.2 Results

Table 6 shows the results of running LangChain SQLQueryChain for GPT-4 and GPT-3.5-turbo-16k over the three sets of LLM-friendly views, all with data samples.

**Table 6** Results for LangChain SQLQueryChain over the industrial database using views [10].

#Line	Model	Accuracy			
		Simple	Medium	Complex	Overall
<b><i>Conceptual schema views</i></b>					
1	GPT-3.5-turbo-16k	0,78	0,18	0,18	0,48
2	GPT-4	<b>0,97</b>	0,55	0,44	0,65
<b><i>Partially extended views</i></b>					
3	GPT-3.5-turbo-16k	0,79	0,55	0,24	<b>0,52</b>
4	GPT-4	0,91	<b>0,76</b>	<b>0,56</b>	<b>0,74</b>
<b><i>Fully extended views</i></b>					
5	GPT-3.5-turbo-16k	0,85	0,39	0,18	0,47
6	GPT-4	0,85	0,61	0,53	0,66

Overall, the accuracy results with GPT-4 were much better than those with GPT-3.5-turbo-16k; if we compare the best accuracy results, GPT-4 achieved an overall accuracy of 74%, whereas GPT-3.5-turbo-16k achieved 52%.

Let us concentrate on the accuracy results with GPT-4. A comparison between the results for the relational schema (line 2 in Table 5) with those for the conceptual schema views (line 2 of Table 6) indicates that the overall accuracy improved from 41% to 65%. This means that simply renaming the tables and columns to terms closer to the end-user vocabulary sufficed to improve accuracy substantially.

Now, a comparison between the results for the conceptual schema views (line 2 of Table 6) with those for the partially extended views (line 4 of Table 6) captures a more subtle improvement. The partially extended views simplify the text-to-SQL task by eliminating joins in certain medium and complex questions. The overall accuracy achieved with these views was substantially better (74% against 65%).

A comparison between the results for the partially extended views (line 4 of Table 6) with those for the fully extended views (line 6 of Table 6) shows a decrease of 8%. Indeed, the fully extended views save more joins, but they require passing much larger view specifications in the prompt. Furthermore, the definition of a fully extended view, which combines several views, requires renaming several columns, which may create columns with similar names. In conjunction – views with many columns and similar column names – confuse the LLM, leading to ambiguous matches with an NL question.

In summary, the results suggest that the partially extended views, with just a few extra columns that predefine joins, is a better alternative than fully extended views, that combine several views. These views also proved to be a much better alternative than using the relational schema or the set of conceptual schema views. From a broader perspective, the accuracy increases when one moves from prompting the LLM with the relational schema to prompting the LLM with LLM-friendly views and data samples.

## 7 Conclusions

Recently, several text-to-SQL tools that explore LLMs emerged that outperformed previous approaches on well-known benchmarks. This article first showed that the performance of a selected set of LLM-based text-to-SQL tools is, however, significantly less when run on two challenging real-world databases, with a large number of tables, columns, and foreign keys. The set included some tools that ranked high in familiar leaderboards, and a model fine-tuned for the text-to-SQL task, with a reported very good performance. A closer analysis reveals that one of the problems lies in that the relational schema is an inappropriate specification of the database from the point of view of the LLM.

The article then introduced database specifications based on LLM-friendly views, that are close to the language of the users' questions and that eliminate frequently used joins, and table row samples. The article proceeds to show that the use of a set of LLM-friendly views and data samples considerably improve the performance of a text-to-SQL prompt strategy over a real-world database. This result suggested that real-world databases require rethinking how schema specifications should be passed to the LLM to recover state-of-the-art text-to-SQL performance without resorting to elaborate (and costly) prompt strategies.

Finally, there is room for further improvement. For example, the LLM-friendly views used in the experiments were created by inspecting the database documentation and mining a log of user questions. Albeit this process was tedious but not too difficult, further work might focus on a tool that automatically creates views on the fly, depending on the NL question submitted, along the lines of the tool described in [7]. Alternatively, one might use a technique based on Retrieval-Augmented Generation [20] to improve the performance of text-to-SQL tools.

## Acknowledgements

This work was partly funded by FAPERJ under grant E-26-204.322/2024; 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.

## Declarations

**Competing Interests:** On behalf of all authors, the corresponding author states that there is no conflict of interest.

**Funding Information:** The funding information is provided in the acknowledgments section.

**Author contribution:** All authors contributed equally and approved the final version of the manuscript.

**Data Availability Statement:** Data is partially available as indicated in Section 5.3.1.

**Research Involving Humans and/or Animals:** Not applicable.

**Informed Consent:** Not applicable.

## References

- [1] 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>
- [2] 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 (2020) <https://doi.org/10.14778/3401960.3401970>
- [3] Affolter, K., Stockinger, K., Bernstein, A.: A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* **28**, 793–819 (2019) <https://doi.org/10.1007/s00778-019-00567-8>
- [4] 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: Riloff, E., Chiang, D., Hockenmaier, J., Tsujii, J. (eds.) *Proc. 2018 Conference on Empirical Methods in Natural Language Processing*, pp. 3911–3921. Association for Computational Linguistics, Brussels, Belgium (2018). <https://doi.org/10.18653/v1/D18-1425> . <https://aclanthology.org/D18-1425>
- [5] Li, J., Hui, B., Qu, G., Yang, J., Li, B., Li, B., Wang, B., Qin, B., Geng, R., Huo, N., Zhou, X., Ma, C., Li, G., Chang, K., Huang, F., Cheng, R., Li, Y.: Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In: *Proceedings of the 37th International Conference on Neural Information Processing Systems. NIPS '23*. Curran Associates Inc., Red Hook, NY, USA (2024)
- [6] Izquierdo, Y.T., García, G.M., Lemos, M., Novello, A., Novelli, B., Damasceno, C., Leme, L.A.P.P., Casanova, M.A.: A platform for keyword search and its application for covid-19 pandemic data. *Journal of Information and Data Management* **12**(5), 521–535 (2021) <https://doi.org/10.5753/jidm.2021.1904>
- [7] Nascimento, E.R., Casanova, M.A., Leme, L.A.P.P., García, G.M., Lemos, M., Izquierdo, Y.T., García, R., Victorio, W.: A family of natural language interfaces for databases based on chatgpt and langchain (short paper). In: *Companion Proceedings of the 42nd International Conference on Conceptual Modeling: Posters and Demos Co-located with ER 2023, Lisbon, Portugal, November 06-09, 2023*. CEUR Workshop Proceedings, vol. 3618 (2023). [https://ceur-ws.org/Vol-3618/pd\\_paper\\_1.pdf](https://ceur-ws.org/Vol-3618/pd_paper_1.pdf)
- [8] Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Chen, L., Lin, J., Lou, D.: C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint* (2023). <https://doi.org/10.48550/arXiv.2307.07306>
- [9] Pourreza, M., Rafiei, D.: Din-sql: decomposed in-context learning of text-to-sql

with self-correction. In: Proceedings of the 37th International Conference on Neural Information Processing Systems. NIPS '23. Curran Associates Inc., Red Hook, NY, USA (2024)

- [10] Nascimento, E.R.S., Garcia, G.M., Feijó, L., Victorio, W., Izquierdo, Y.T., Oliveira, A., Coelho, G.M.C., Lemos, M., Garcia, R.L.S., Leme, L.A.P.P., Casanova, M.A.: Text-to-sql meets the real-world. In: Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1: ICEIS, pp. 61–72. SciTePress, Setúbal, Portugal (2024). <https://doi.org/10.5220/0012555200003690> . INSTICC
- [11] Nascimento, E.R.S., Izquierdo, Y.T., Garcia, G.M., Coelho, G., Feijó, L., Lemos, M., Leme, L.A.P.P., Casanova, M.A.: My database user is a large language model. In: Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1: ICEIS, pp. 800–806. SciTePress, Setúbal, Portugal (2024). <https://doi.org/10.5220/0012697700003690> . INSTICC
- [12] Zhong, V., Xiong, C., R., S.: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint (2017). <https://doi.org/10.48550/arXiv.1709.00103>
- [13] Guo, J., Si, Z., Wang, Y., Liu, Q., Fan, M., Lou, J.-G., Yang, Z., Liu, T.: Chase: A large-scale and pragmatic chinese dataset for cross-database context-dependent text-to-sql. In: Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, pp. 2316–2331 (2021). Available at: <https://aclanthology.org/2021.acl-long.180>
- [14] Ping, W.J.: Open-sourcing SQLEval: our framework for evaluating LLM-generated SQL (2023). <https://defog.ai/blog/open-sourcing-sqlevel/>
- [15] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., Zhou, J.: Text-to-sql empowered by large language models: A benchmark evaluation. arXiv preprint (2023). <https://doi.org/10.48550/arXiv:2308.15363>
- [16] Izquierdo, Y.T., García, G.M., Menendez, E.S., Casanova, M.A., Dartayre, F., Levy, C.H.: Quiow: a keyword-based query processing tool for rdf datasets and relational databases. In: Hartmann, S., Ma, H., Hameurlain, A., Pernul, G., Wagner, R.R. (eds.) International Conference on Database and Expert Systems Applications (DEXA), pp. 259–269. Springer, Cham (2018). [https://doi.org/10.1007/978-3-319-98812-2\\_22](https://doi.org/10.1007/978-3-319-98812-2_22)
- [17] García, G.M., Izquierdo, Y.T., Menendez, E., Dartayre, F., Casanova, M.A.: Rdf keyword-based query technology meets a real-world dataset. In: Proceedings of the 20th International Conference on Extending Database Technology (EDBT), pp. 656–667. OpenProceedings.org, Venice, Italy (2017). <https://doi.org/10.5441/002/edbt.2017.86>

- [18] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., Zhou, D.: Chain-of-thought prompting elicits reasoning in large language models. arXiv preprint (2023). <https://doi.org/10.48550/arXiv.2310.12516>
- [19] Yu, X., Cheng, H., Liu, X., Roth, D., Gao, J.: Automatic hallucination assessment for aligned large language models via transferable adversarial attacks. arXiv preprint (2023). <https://doi.org/10.48550/arXiv:2310.12516>
- [20] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., Kiela, D.: 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., Red Hook, NY, USA (2020). [https://proceedings.neurips.cc/paper\\_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf](https://proceedings.neurips.cc/paper_files/paper/2020/file/6b493230205f780e1bc26945df7481e5-Paper.pdf)