








# On the Text-to-SQL Task Supported by Database Keyword Search

Eduardo R. Nascimento<sup>1</sup><sup>a</sup>, Caio Viktor S. Avila<sup>1,4</sup><sup>b</sup>, Yenier T. Izquierdo<sup>1</sup><sup>c</sup>, Grettel M. García<sup>1</sup><sup>d</sup>,  
Lucas Feijó L. Andrade<sup>1</sup><sup>e</sup>, Michelle S.P. Facina<sup>2</sup>, Melissa Lemos<sup>1</sup><sup>f</sup>, Marco A. Casanova<sup>1,3</sup><sup>g</sup>

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

<sup>2</sup> Petrobras, Rio de Janeiro, RJ, Brazil CEP 20231-030

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

<sup>4</sup> Departamento de Computação, UFC, Fortaleza, Brazil, CEP 60440-900

{rogerrsn, ytorres, ggarcia, lucasfeijo, melissa}@tecgraf.puc-rio.br, caioviktor@alu.ufc.br, michelle@petrobras.com.br, casanova@inf.puc-rio.br

Keywords: Text-to-SQL, Database Keyword Search, Large Language Models, Relational Databases.

Abstract: Text-to-SQL prompt strategies based on Large Language Models (LLMs) achieve remarkable performance on well-known benchmarks. However, when applied to real-world databases, their performance is significantly less than for these benchmarks, especially for Natural Language (NL) questions requiring complex filters and joins to be processed. This paper then proposes a strategy to compile NL questions into SQL queries that incorporates a dynamic few-shot examples strategy and leverages the services provided by a database keyword search (KwS) platform. The paper details how the precision and recall of the schema-linking process are improved with the help of the examples provided and the keyword-matching service that the KwS platform offers. Then, it shows how the KwS platform can be used to synthesize a view that captures the joins required to process an input NL question and thereby simplify the SQL query compilation step. The paper includes experiments with a real-world relational database to assess the performance of the proposed strategy. The experiments suggest that the strategy achieves an accuracy on the real-world relational database that surpasses state-of-the-art approaches. The paper concludes by discussing the results obtained.

## 1 INTRODUCTION


The *Text-to-SQL* task is defined as “given a relational database  $D$  and a natural language (NL) sentence  $Q_N$  that describes a question on  $D$ , generate an SQL query  $Q_{SQL}$  over  $D$  that expresses  $Q_N$ ” (Katsogiannis-Meimarakis and Koutrika, 2023; Kim et al., 2020).


Numerous tools have addressed this task with relative success (Affolter et al., 2019; Katsogiannis-Meimarakis and Koutrika, 2023; Kim et al., 2020; Shi et al., 2024) over well-known benchmarks, such as Spider – Yale Semantic Parsing and Text-to-SQL Challenge (Yu et al., 2018) and BIRD – BIG Bench for LaRge-scale Database Grounded Text-to-SQL Eval-


uation (Li et al., 2024). The leaderboards of these benchmarks point to a firm trend: the best text-to-SQL tools are all based on Large Language Models (LLMs) (Shi et al., 2024).


Text-to-SQL tools must face several challenges, as a consequence of the complexity of SQL (Yu et al., 2018) and of the database itself. In particular, *real-world databases* raise challenges for several reasons, among which: (1) The relational schema is often large; (2) The relational schema is often an inappropriate specification of the database from the point of view of the LLM; (3) The data semantics are often complex; (4) Metadata and data are often ambiguous. Indeed, the performance of some of the best LLM-based text-to-SQL tools on real-world databases is significantly less than that observed for the Spider and BIRD benchmarks (Nascimento et al., 2024a; Lei et al., 2024).


This paper then addresses the *real-world text-to-SQL problem*, which is the version of the text-to-SQL problem for real-world databases. Albeit the original problem has been investigated for some time, this


<sup>a</sup>  <https://orcid.org/0009-0005-3391-7813>

<sup>b</sup>  <https://orcid.org/0009-0002-3899-0014>

<sup>c</sup>  <https://orcid.org/0000-0003-0971-8572>

<sup>d</sup>  <https://orcid.org/0000-0001-9713-300X>

<sup>e</sup>  <https://orcid.org/0009-0006-4763-8564>

<sup>f</sup>  <https://orcid.org/0000-0003-1723-9897>

<sup>g</sup>  <https://orcid.org/0F000-0003-0765-9636>

version is considered far from solved, as argued in (Floratos et al., 2024; Lei et al., 2024).

The first contribution of the paper is a novel strategy to compile NL questions into SQL queries that leverages the services provided by a database keyword search (KwS) platform, called DANKE (Izquierdo et al., 2021; Izquierdo et al., 2024). The proposed strategy is the first one to explore a symbiotic combination of a KwS platform and a prompt strategy to process NL questions.

The second contribution of the paper is a set of experiments with a real-world benchmark to assess the performance of the proposed strategy. The benchmark is built upon a relational database with a challenging schema, which is in production at an energy company, and a set of 100 NL questions carefully defined to reflect the NL questions users submit and to cover a wide range of SQL constructs (Spider and BIRD, two of the familiar text-to-SQL benchmarks, were not adopted for the reasons explained in Section 2.1). These new results, combined with results from (Nascimento et al., 2024a), indicate that the proposed strategy performs significantly better on the real-world benchmark than LangChain SQLQueryChain, SQLCoder<sup>1</sup>, “C3 + ChatGPT + Zero-Shot” (Dong et al., 2023), and “DIN-SQL + GPT-4” (Pourreza and Rafiei, 2024).

The paper is organized as follows. Section 2 covers related work. Section 3 describes the database keyword search platform adopted in the paper. Section 4 details the proposed text-to-SQL strategy. Section 5 presents the experiments, including the real-world benchmark used. Finally, Section 6 contains the conclusions.

## 2 RELATED WORK

### 2.1 Text-to-SQL Datasets

The Spider – Yale Semantic Parsing and Text-to-SQL Challenge (Yu et al., 2018) defines 200 datasets, covering 138 different domains, for training and testing text-to-SQL tools. Most databases in Spider have very small schemas – the largest five databases have between 16 and 25 tables, and about half have schemas with five tables or fewer. Furthermore, all Spider NL questions are phrased in terms used in the database schemas. These two limitations considerably reduce the difficulty of the text-to-SQL task. Therefore, the results reported in the Spider leaderboard are biased toward databases with small schemas and NL ques-

tions written in the schema vocabulary, which is not what one finds in real-world databases.

BIRD – BIG Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation (Li et al., 2024) 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. Again, all NL questions are phrased in the terms used in the database schemas.

Despite the availability of these benchmark datasets for the text-to-SQL task, and inspired by them, Section 5.1 describes a benchmark dataset constructed specifically to test strategies designed for the real-world text-to-SQL task. The benchmark dataset consists of a relational database and a set of 100 test NL questions and their ground-truth SQL translations. The database schema is inspired by a real-world schema and is far more challenging than most of the database schemas available in Spider or BIRD. The database is populated with real data with a semantics which is sometimes not easily mapped to the semantics of the terms the users adopt (such as “criticality\_level = 5” encodes “critical orders”), which is a challenge for the text-to-SQL task not captured by unpopulated databases, as in Spider. Finally, the NL questions mimic those posed by real users, and cover a wide range of SQL constructs.

### 2.2 Text-to-SQL Tools

A comprehensive survey of text-to-SQL strategies can be found in (Shi et al., 2024), including a discussion of benchmark datasets, prompt engineering, and fine-tuning methods, partly covered in what follows.

Several text-to-SQL tools were tested in (Nascimento et al., 2024a) against the benchmark used in this paper – SQLCoder, LangChain SQLQueryChain, C3, and DIN+SQL. Despite the impressive results of C3 and DIN on Spider, and of SQLCoder on a specific benchmark, the performance of these tools on the benchmark used in this paper was significantly lower (Nascimento et al., 2024a), and much less than that of the strategy described in Section 4. A similar remark applies to LangChain SQLQueryChain, whose results are shown in Line 1 of Table 2.

Lastly, a text-to-SQL tool that leverages dynamic few-shot examples was introduced in (Coelho et al., 2024). Line 4 of Table 2 shows the results the tool achieved on the benchmark used in this paper.

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

### 3 A DATABASE KEYWORD QUERY PROCESSING TOOL

DANKE is the keyword search platform currently deployed for the industrial database described in Section 5.1 and used for the experiments. The reader is referred to (Izquierdo et al., 2021; Izquierdo et al., 2024) for the details of the platform.

DANKE operates over both relational databases and RDF datasets, and is designed to compile a keyword query into an SQL or SPARQL query that returns the best data matches. For simplicity, the description that follows uses the relational terminology.

DANKE’s architecture comprises three main components: (1) *Storage Module*; (2) *Preparation Module*; and (3) *Data and Knowledge Extraction Module*.

The *Storage Module* houses a centralized relational database, constructed from various data sources. The database is described by a *conceptual schema*, treated in what follows as a relational schema, again for simplicity.

The *Storage Module* also holds the data indices required to support the keyword search service. The indexing process is enriched to create a *keyword dictionary*.

The *Preparation Module* has tools for creating the conceptual schema and for constructing and updating the centralized database through a pipeline typical of a data integration process. The conceptual schema is defined by de-normalizing the relational schemas of the underlying databases and indicating which columns will have their values indexed.

The *Data and Knowledge Extraction Module* has two main sub-modules, *Query Compilation* and *Query Processing*.

Let  $R$  be the referential dependencies diagram of the database schema in question, where the nodes of  $R$  are the tables and there is an edge between nodes  $t$  and  $u$  iff there is a foreign key from  $t$  to  $u$  or vice-versa. Given a set of keywords  $K$ , let  $T_K$  be a set of table schemes whose instances match the largest set of keywords in  $K$ . The query compilation sub-module first constructs a Steiner tree  $S_K$  of  $R$  whose end nodes are the set  $T_K$ . This is the central point since it guarantees that the final SQL query will not return unconnected data, as explored in detail in (García et al., 2017). If  $R$  is connected, then it is always possible to construct one such Steiner tree; otherwise, one would have to find a Steiner forest to cover all tables in  $T_K$ .

Using the Steiner tree, the Query Compilation sub-module compiles the keyword query into an SQL query that includes restriction clauses representing the keyword matches and join clauses connecting the restriction clauses. Without such join clauses,

an answer would be a disconnected set of tuples, which hardly makes sense. The generation of the join clauses uses the Steiner tree edges.

Lastly, DANKE’s internal API was expanded to support the text-to-SQL strategy described in Section 4. Briefly, it now offers the following services:

- *Keyword Match Service*: receives a set  $K$  of keywords and returns the set  $K_M$  of pairs  $(k, d_k)$  such that  $k \in K$  and  $d_k$  is the dictionary entry that best matches  $k$ . The dictionary entry  $d_k$  will be called the *data associated* with  $k$ .
- *View Synthesis Service*: receives a set  $S'$  of tables and returns a view  $V$  that best joins all tables in  $S'$ , using the Steiner tree optimization heuristic mentioned above.

### 4 A TEXT-TO-SQL STRATEGY

The proposed text-to-SQL strategy comprises two modules, *schema linking* and *SQL query compilation*. The two modules use a dynamic few-shot examples strategy that retrieves a set of samples from a *synthetic dataset*. The key point is the use of services provided by DANKE to enhance schema linking and simplify SQL query compilation. In particular, DANKE will generate a single SQL view containing all data and encapsulating all joins necessary to answer the input NL question.

The *synthetic dataset*  $D$  for the database  $DB$  contains pairs  $(Q_N, Q_{SQL})$ , where  $Q_N$  is an NL question and  $Q_{SQL}$  is its SQL translation. Such pairs should provide examples that help the LLM understand how the database schema is structured, how the user’s terms map to terms of the database schema, and how NL language constructions map to data values. The synthetic dataset construction process is described in detail in (Coelho et al., 2024).

Let  $DB$  be a relational database with schema  $S$  and  $D$  be the synthetic dataset created for  $DB$ . Let  $Q_N$  be an NL question over  $S$ .

The schema linking module finds a minimal set  $S' \subset S$  such that  $S'$  has all tables in  $S$  required to answer  $Q_N$ . It has the following components (see Figure 1): *Keyword Extraction and Matching*; *Dynamic Few-shot Examples Retrieval (DFE)*; *Schema Linking*.

*Keyword Extraction and Matching* receives  $Q_N$  as input; calls the LLM to extract a set  $K$  of keywords from  $Q_N$ ; calls the DANKE Keyword Matching service to match  $K$  with the dictionary, creating a final set  $K_M$  of keywords and associated data.

*Dynamic Few-shot Examples Retrieval (DFE)* receives  $Q_N$  as input; retrieves from the synthetic

dataset  $D$  a set of  $k$  examples whose NL questions are most similar to  $Q_N$ , generating a list  $T = [(Q_1, F_1), \dots, (Q_k, F_k)]$ , where  $F_i$  is the set of tables in the FROM clause of the SQL query associated with  $Q_i$  in  $D$ .

*Schema Linking* receives as input  $Q_N$ ,  $K_M$ , and  $T$ ; retrieves the set of tables in  $S$  and their columns; calls the LLM to create  $S'$  prompted by  $Q_N$ ,  $K_M$ ,  $S$ , and  $T$ ; returns  $S'$  and  $K_M$ .

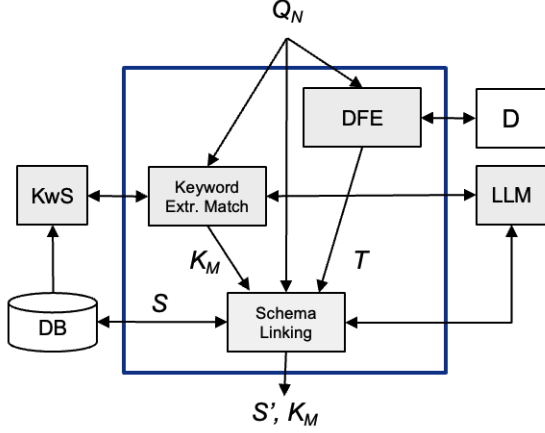


Figure 1: Schema linking module.

The SQL query compilation module receives as input the NL question  $Q_N$ , the set of tables  $S'$ , and the set  $K_M$  of keywords and associated data, and returns an SQL query  $Q_{SQL}$ . It has the following major components (see Figure 2): *View Synthesis*; *Question Decomposition*; *Dynamic Few-shot Examples Retrieval (DFE)*; *SQL Compilation*.

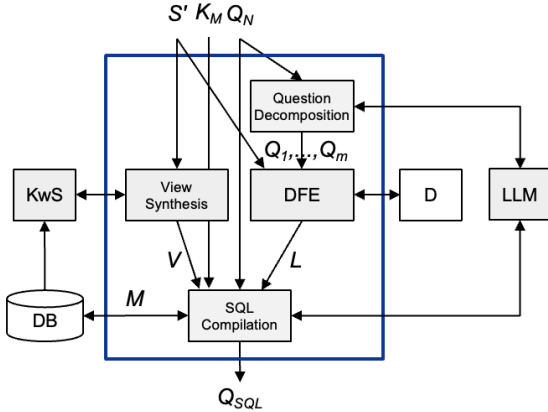


Figure 2: SQL query compilation module.

*View Synthesis* receives the set of tables  $S'$  as input; calls the DANKE View Synthesis service to synthesize a view  $V$  that joins the tables in  $S'$ ; returns  $V$ .

*Question Decomposition* receives  $Q_N$  as input; de-

composes  $Q_N$  into sub-questions  $Q_1, \dots, Q_m$ ; returns  $Q_1, \dots, Q_m$ .

*Dynamic Few-shot Examples Retrieval (DFE)* receives the list  $Q_1, \dots, Q_m$  as input; for each  $i \in [1, m]$ , retrieves from the synthetic dataset  $D$  a set of  $p$  examples whose NL questions are most similar to  $Q_i$  and whose SQL queries are over  $S'$ , generating a list  $L_i = [(Q_{i_1}, SQL_{i_1}), \dots, (Q_{i_p}, SQL_{i_p})]$  in decreasing order of similarity of  $Q_{i_j}$  to  $Q_i$ ; creates the final list  $L$ , with  $k$  elements, by intercalating the lists  $L_i$  and retaining the top- $k$  pairs.

*SQL Compilation* receives as input  $Q_N$ ,  $V$ ,  $K_M$ , and  $L$ ; retrieves from  $DB$  a set  $M$  of row samples of  $V$ ; calls the LLM to compile  $Q_N$  into an SQL query  $Q_{SQL}$  over  $V$ , when prompted with  $Q_N$ ,  $V$ ,  $K_M$ ,  $M$  and  $L'$ ; returns  $Q_{SQL}$ .

## 5 EXPERIMENTS

### 5.1 A Benchmark Dataset

This section describes a benchmark to help investigate the real-world text-to-SQL task. The benchmark consists of a relational database, a set of 100 test NL questions and their SQL *ground-truth* translations, and a set of *partially extended views*.

The selected database is a real-world relational database that stores data related to the integrity management of an energy company’s industrial assets. The relational schema of the adopted database contains 27 relational tables with, in total, 585 columns and 30 foreign keys (some multi-column), where the largest table has 81 columns.

The benchmark contains a set of 100 NL questions that consider the terms and questions experts use when requesting information related to the maintenance and integrity processes. The ground-truth SQL queries were manually defined so that the execution of a ground-truth SQL query returns the expected answer to the corresponding NL question.

An NL question is classified into *simple*, *medium*, and *complex*, based on the complexity of its corresponding ground-truth SQL query, as in the Spider benchmark. The set contains 33 simple, 33 medium, and 34 complex NL questions.

Lastly, table and column names in the relational schema use terms based on an internal naming convention for database objects, which makes it hard for end-users, including non-human users such as LLMs, to directly use the relational schema. To address this fact, the benchmark also includes a set of *partially extended views* (Nascimento et al., 2024b) that rename table and column names of the relational schema to

end users’ terms. Such views also have new columns that pre-define joins that follow foreign keys and import selected columns from the referenced tables to facilitate SQL query compilation.

## 5.2 Evaluation Procedure

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 predicted SQL query may be compared with the corresponding ground-truth SQL query based on the results they return. The results of the automated procedure were manually checked to eliminate false positives and false negatives. The reader is referred to (Nascimento et al., 2024a) for the details.

## 5.3 Experiments with Schema Linking

### 5.3.1 Experimental Setup

The first set of experiments evaluated several alternatives for performing the schema linking task.

The experiments adopted the benchmark described in Section 5.1. For each NL question, the ground-truth minimum sets of tables necessary to answer the NL question is the set of tables in the FROM clause of the ground-truth SQL query. The experiments used GPT-3.5 Turbo and GPT-4, but only the results obtained with GPT-4 were noteworthy.

Table 1 presents the results of the experiments for the following alternatives:

1. (*LLM*): A strategy that prompts an LLM with  $Q_N$  and  $S$  to find the set of tables  $S'$ .
2. (*DANKE*): A strategy that, first, uses DANKE to extract a set of keywords  $K$  from  $Q_N$ , and then extracts the set of tables  $S'$  from the information associated with  $K$  in DANKE’s dictionary.
3. (*LLM+DFE*): A strategy that, first, finds a set of examples  $T$  from  $D$  using  $Q_N$ , and then prompts an LLM with  $Q_N$ ,  $S$ , and  $T$  to find the set of tables  $S'$ .
4. (*LLM+DANKE*): A strategy that, first, uses an LLM to extract a set  $K$  of keywords from  $Q_N$ , and then extracts the set of tables  $S'$  from the information associated with  $K$  in DANKE’s dictionary.
5. (*LLM+DANKE+DFE*): A strategy that, first, uses an LLM to extract a set  $K$  of keywords from  $Q_N$ , retrieves the information associated with  $K$  from DANKE’s dictionary, creating a set  $K_M$ , finds a set of examples  $T$  from  $D$  using  $Q_N$ , and then prompts

an LLM with  $Q_N$ ,  $S$ ,  $K_M$ , and  $T$  to find the set of tables  $S'$ .

6. (*Complete*): The entire Schema Linking process.

### 5.3.2 Results

Table 1 presents the precision, recall, and F1-score for the experiments using the Schema Linking process. Briefly, the results show that:

1. (*LLM*): Alternative 1 obtained an F1-score of 0.851. It had a performance poorer than Alternative 2, which used just DANKE.
2. (*DANKE*): Alternative 2 obtained an F1-score of 0.900. Note that DANKE achieved a better result than Alternatives 1 and 3 (which do not use DANKE), although DANKE does no syntactic or semantic processing of the user question  $Q_N$ , and may incorrectly match terms in  $Q_N$  to terms in the database schema or to data values.
3. (*LLM+DFE*): Alternative 3 obtained an F1-score of 0.868. The use of DFE improved the results achieved by Alternative 1, but the results were still lower than those of Alternative 2.
4. (*LLM+DANKE*): Alternative 4 increased the F1-score to 0.930. Enriching the prompt with the keywords extracted by DANKE from  $Q_N$  yielded consistent improvements in both precision and recall. This is due to DANKE’s ability to find references to column values, associating them with the table/column where the value occurs. This feature allowed DANKE to find implicit references that were previously impossible for the LLM to discover since it had no knowledge about the database instances.
5. (*LLM+DANKE+DFE*): Alternative 5 increased the F1-score to 0.950.
6. (*Complete – GPT-4*): The complete Schema Linking process achieved an F1-Score of 0.996, the best result. Using the LLM to extract keywords from  $Q_N$  improved the results of DANKE. Although DANKE may still return incorrect terms, the LLM corrects them.
7. (*Complete – GPT-4o*): Using GPT-4o resulted in a slight decrease in the F1-score to 0.995.

In general, these results show that DANKE, together with the LLM, performed effectively in the Schema Linking process for NL questions. Considering that the complete Schema Linking process achieved a recall of 1.0, it returned all tables required to answer each NL question. Thus, the Schema Linking process does not impact the SQL Query Compilation step, although the extra tables may create distractions for the LLM (see Section 5.4).

Table 1: Results for the schema linking alternatives (all with GPT-4, except the last line).

#	Method	Precision	Recall	F1-score
1	LLM	0.864	0.886	0.851
2	DANKE	0.860	0.983	0.900
3	LLM+DFE	0.940	0.843	0.868
4	LLM+DANKE	0.930	0.930	0.930
5	LLM+DFE+DANKE	0.993	0.983	0.950
6	Complete – GPT-4	0.993	<b>1.000</b>	<b>0.996</b>
7	Complete – GPT-4o	<b>1.000</b>	0.995	0.995

## 5.4 Experiments with SQL Query Compilation

### 5.4.1 Experimental Setup

The experiments were based on LangChain SQL-QueryChain, 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.

The experiments executed the 100 questions introduced in Section 5.1 in nine alternatives:

1. (*Relational Schema*): SQLQueryChain executed over the relational schema of the benchmark database.
2. (*Partially Extended Views*): SQLQueryChain executed over the partially extended views of the benchmark database.
3. (*Partially Extended Views and DFE*): SQL-QueryChain executed over the partially extended views using only the DFE technique.
4. (*Partially Extended Views, DFE, and Question Decomposition*): SQLQueryChain executed over the partially extended views, using Question Decomposition and the DFE technique.
5. (*The Proposed Text-to-SQL Strategy – GPT-4*): The proposed Text-to-SQL Strategy, using GPT-4-32K.
6. (*The Proposed Text-to-SQL Strategy – GPT-4o*): The proposed Text-to-SQL Strategy, using GPT-4o.
7. (*The Proposed Text-to-SQL Strategy – LLaMA 3.1-405B-Instruct*): The proposed Text-to-SQL Strategy, using LLaMA 3.1-405B-Instruct.
8. (*The Proposed Text-to-SQL Strategy – Mistral Large*): The proposed Text-to-SQL Strategy, using Mistral Large.
9. (*The Proposed Text-to-SQL Strategy – Claude 3.5-Sonnet*): The proposed Text-to-SQL Strategy, using Claude 3.5-Sonnet.

Alternatives 1–6 ran on the OpenAI platform, and Alternatives 7–9 on the AWS Bedrock platform.

### 5.4.2 Results

Table 2 summarizes the results for the various alternatives. Columns under “#Correct Predicted Questions” show the number of NL questions per type correctly translated to SQL (recall that there are 33 simple, 33 medium, and 34 complex NL questions, with a total of 100); columns under “Accuracy” indicate the accuracy results per NL question type and the overall accuracy; the last column shows the total elapsed time to run all 100 NL questions.

The results for Alternatives 1, 2, and 3 were reported in (Nascimento et al., 2024a; Nascimento et al., 2024b; Coelho et al., 2024), respectively. They are repeated in Table 2 for comparison with the results of this paper.

The results for Alternative 4 show that Question Decomposition produced an improvement in total accuracy from 0.79 to 0.84. This reflects the diversity of examples passed to the LLM when they are retrieved for each sub-question, as already pointed out in (Oliveira et al., 2025).

The results for Alternative 5 show that the key contribution of this paper, the text-to-SQL strategy described in Section 4, indeed leads to a significant improvement in the total accuracy for the case study database, as well as the accuracies for the medium and complex NL questions.

The results for Alternative 6 indicate a slight decrease in the total accuracy to 0.90 when GPT-4o is adopted, possibly due to the non-deterministic behavior of the models. However, while GPT-4-32K took 17 minutes to run all 100 questions, GPT-4o took only 7 minutes.

The results for Alternatives 7–9 show a decrease in the total accuracy to 0.81%, 0.77%, and 0.74%, respectively, with a much higher total elapsed time, when compared with GPT-4o, but comparable to that of GPT-4-32K.

### 5.4.3 Discussion

The results in Table 2 show that the proposed strategy (Line 5) correctly processed four more medium NL questions and six more NL complex questions than the previous best strategy (Line 4). However, these results hide the fact that the proposed strategy processed four complex NL questions that none of the strategies previously tested on the same database and set of questions have correctly handled, including C3 and DIN.

Table 2: Summary of the results.

#Line	Alternatives / Model	#Samples	#Correct Predicted Questions				Accuracy				Total Elapsed Time
			Simple	Medium	Complex	Total	Simple	Medium	Complex	Total	
1	<i>Relational Schema</i>										
	GPT-4	-	22	11	8	41	0,67	0,33	0,23	0,41	23min
2	<i>Partially Extended Views</i>										
	GPT-4	-	30	25	19	74	0,91	0,76	0,56	0,74	N/A
3	<i>Partially Extended Views and DFE</i>										
	GPT-4-32K	Top 8	32	24	23	79	0,97	0,73	0,68	0,79	11min
4	<i>Partially Extended Views, DFE, and Question Decomposition</i>										
	GPT-4-32K	Top 8	32	28	24	84	0,97	0,85	0,71	0,84	31min
5	<i>The Proposed Text-to-SQL Strategy</i>										
	GPT-4-32K	Top 8	31	32	30	93	0,94	0,97	0,98	<b>0,93</b>	17min
6	<i>The Proposed Text-to-SQL Strategy</i>										
	GPT-4o	Top 8	30	30	30	90	0,91	0,91	0,88	0,90	7min
7	<i>The Proposed Text-to-SQL Strategy</i>										
	LLaMA 3.1-405B-Instruct	Top 8	25	28	28	81	0,76	0,85	0,82	0,81	19min
8	<i>The Proposed Text-to-SQL Strategy</i>										
	Mistral Large	Top 8	26	27	24	77	0,79	0,82	0,71	0,77	16min
9	<i>The Proposed Text-to-SQL Strategy</i>										
	Claude 3.5-Sonnet	Top 8	24	24	26	74	0,73	0,73	0,76	0,74	17min

As for the other models – Llama 3.1-405B Instruct, Mistral Large, and Claude 3.5 Sonnet – the most common source of error was the use of the CONTAINS function, which requires the target column to be indexed, but this was not always the case; the correct filter would have to use LIKE.

## 6 CONCLUSIONS

This paper proposed a text-to-SQL strategy that leverages the services provided by DANKE, a database keyword search platform.

The paper detailed how the schema-linking process can be improved with the help of the keyword extraction service that DANKE provides. Then, it showed how DANKE can be used to synthesize a view that captures the joins required to process an input NL question and thereby simplify the SQL query compilation step.

The paper included experiments with a real-world relational database to assess the performance of the proposed strategy. The results in Section 5.3 showed that the precision and recall of the schema-linking process indeed improved with the help of the keyword extraction service that DANKE provides. The discussion in Section 5.4 suggested that creating a view with the help of DANKE also helped with the SQL query compilation process. In conjunction, these results indicated that the proposed strategy achieved a total accuracy in excess of 90% over a benchmark built upon a relational database with a challenging schema and a set of 100 questions carefully defined to reflect the

questions users submit and to cover a wide range of SQL constructs. The total accuracy was much higher than that achieved by SQLCoder, LangChain SQL-QueryChain, C3, and DIN+SQL on the same benchmark, as reported in (Nascimento et al., 2024a).

As for future work, the proposed strategy should be tested and compared against other strategies using additional databases and test questions. A second demand is to address the problem that Natural Language questions are intrinsically ambiguous. DANKE’s matching process helps but should be complemented with a different approach, perhaps incorporating the user in a disambiguation loop.

## ACKNOWLEDGEMENTS

This work was partly funded by FAPERJ under grant E-26/204.322/2024; by CNPq under grant 302303/2017-0; and by Petrobras, under research agreement 2022/00032-9 between CENPES and PUC-Rio.

## REFERENCES

- Affolter, K., Stockinger, K., and Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28:793–819.
- Coelho, G., Nascimento, E. S., Izquierdo, Y., García, G., Feijó, L., Lemos, M., Garcia, R., de Oliveira, A., Pinheiro, J., and Casanova, M. (2024). Improving the accuracy of text-to-sql tools based on large language models for real-world relational databases. In Strauss,

- C., Amagasa, T., Manco, G., Kotsis, G., Tjoa, A., and Khalil, I., editors, *Database and Expert Systems Applications*, pages 93–107, Cham. Springer Nature Switzerland.
- Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Chen, L., Lin, J., and Lou, D. (2023). C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2307.07306>.
- Floratou, A. et al. (2024). Nl2sql is a solved problem... not! In *Conference on Innovative Data Systems Research*.
- García, G., Izquierdo, Y., Menendez, E., Dartayre, F., and Casanova, M. (2017). Rdf keyword-based query technology meets a real-world dataset. In *Proceedings of the 20th International Conference on Extending Database Technology (EDBT)*, pages 656–667, Venice, Italy. OpenProceedings.org.
- Izquierdo, Y., García, G., Lemos, M., Novello, A., Novelli, B., Damasceno, C., Leme, L., and Casanova, M. (2021). A platform for keyword search and its application for covid-19 pandemic data. *Journal of Information and Data Management*, 12(5):521–535.
- Izquierdo, Y., Lemos, M., Oliveira, C., Novelli, B., García, G., Coelho, G., Feijó, L., Coutinho, B., Santana, T., Garcia, R., and Casanova, M. (2024). Busca360: A search application in the context of top-side asset integrity management in the oil & gas industry. In *Anais do XXXIX Simpósio Brasileiro de Bancos de Dados*, pages 104–116, Porto Alegre, RS, Brasil. SBC.
- Katsogiannis-Meimarakis, G. and Koutrika, G. (2023). A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.
- Kim, H., So, B.-H., Han, W.-S., and Lee, H. (2020). Natural language to sql: Where are we today? *Proc. VLDB Endow.*, 13(10):1737–1750.
- Lei, F. et al. (2024). Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2411.07763>.
- 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., and Li, Y. (2024). 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*, Red Hook, NY, USA. Curran Associates Inc.
- Nascimento, E., García, G., Feijó, L., Victorio, W., Izquierdo, Y., Oliveira, A., Coelho, G., M., L., Garcia, R., Leme, L., and Casanova, M. (2024a). Text-to-sql meets the real-world. In *Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1: ICEIS*, pages 61–72, Setúbal, Portugal. INSTICC, SciTePress.
- Nascimento, E., Izquierdo, Y., García, G., Coelho, G., Feijó, L., Lemos, M., Leme, L., and M.A., C. (2024b). My database user is a large language model. In *Proceedings of the 26th International Conference on Enterprise Information Systems - Volume 1: ICEIS*, pages 800–806, Setúbal, Portugal. INSTICC, SciTePress.
- Oliveira, A., Nascimento, E., Pinheiro, J., Avila, C., Coelho, G., Feijó, L., Izquierdo, Y., García, G., Leme, L., Lemos, M., and Casanova, M. (2025). Small, medium, and large language models for text-to-sql. In Maass, W., Han, H., Yasar, H., and Multari, N., editors, *Conceptual Modeling*, pages 276–294, Cham. Springer Nature Switzerland.
- Pourreza, M. and Rafiei, D. (2024). 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*, Red Hook, NY, USA. Curran Associates Inc.
- Shi, L., Tang, Z., Zhang, N., Zhang, X., and Yang, Z. (2024). A survey on employing large language models for text-to-sql tasks. *arXiv preprint*. <https://doi.org/10.48550/arXiv.2407.15186>.
- Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018). 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., and Tsujii, J., editors, *Proc. 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.