Text-to-SQL Meets the Real-World

Eduardo R. Nascimento[1,4][a], Grettel M. García[1][b], Lucas Feijó[1][c], Wendy Z. Victorio[1,4][d], Yenier T. Izquierdo[1][e], Aiko R. de Oliveira[4][f], Gustavo M. C. Coelho[1][g], Melissa Lemos[1,4][h], Robinson L. S. Garcia[2][i], Luiz A. P. Paes Leme[3][j] and Marco A. Casanova[1,4][k]

[1]*Instituto Tecgraf, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil*

[2]*Petrobras, Rio de Janeiro, 20031-912, RJ, Brazil*

[3]*Instituto de Computação, UFF, Niterói, 24210-310, RJ, Brazil*

[4]*Departamento de Informática, PUC-Rio, Rio de Janeiro, 22451-900, RJ, Brazil*

Keywords: Text-to-SQL, GPT, Large Language Models, Industrial Databases.

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*". Numerous tools have addressed this task with relative success over well-known benchmarks. 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. When adopted for industrial-size databases, with a large number of tables, columns, and foreign keys, the performance of LLM-based text-to-SQL tools is, however, significantly less than that reported for the benchmarks. This paper then investigates how a selected set of LLM-based text-to-SQL tools perform over two challenging databases, an openly available database, Mondial, and a proprietary industrial database. The paper also proposes a new LLM-based text-to-SQL tool that combines features from tools that performed well over the Spider and BIRD benchmarks. Then, the paper describes how the selected tools and the proposed tool, running under GPT-3.5 and GPT-4, perform over the Mondial and the industrial databases over a suite of 100 carefully defined natural language questions that are closely related to those observed in practice. It concludes with a discussion of the results obtained.

1 INTRODUCTION

Text-to-SQL (Katsogiannis-Meimarakis and Koutrika, 2023), also referred to as NL2SQL (Kim et al., 2020), 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 (Affolter et al., 2019)(Katsogiannis-Meimarakis and Koutrika, 2023)(Kim et al., 2020) 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 Evaluation (Li et al., 2023). 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. One of the reasons is that such databases have large schemas, whereas these benchmarks feature a large number of databases whose schemas are quite small. Indeed, using just the number of tables, of the nearly 200 databases used for training in Spider, the largest one has 25 tables (and it is the only one with more than 20 tables) and, of the 20 used for testing, the largest one has 11 tables. The BIRD benchmark tries

[a] https://orcid.org/0009-0005-3391-7813
[b] https://orcid.org/0000-0001-9713-300X
[c] https://orcid.org/0009-0006-4763-8564
[d] https://orcid.org/0009-0003-0545-2612
[e] https://orcid.org/0000-0003-0971-8572
[f] https://orcid.org/0009-0001-8970-0454
[g] https://orcid.org/0000-0003-2951-4972
[h] https://orcid.org/0000-0003-1723-9897
[i] https://orcid.org/0000-0002-0528-5151
[j] https://orcid.org/0000-0001-6014-7256
[k] https://orcid.org/0000-0003-0765-9636

Nascimento, E., García, G., Feijó, L., Victorio, W., Izquierdo, Y., R. de Oliveira, A., Coelho, G., Lemos, M., Garcia, R., Leme, L. and Casanova, M.
Text-to-SQL Meets the Real-World.
DOI: 10.5220/0012555200003690

In *Proceedings of the 26th International Conference on Enterprise Information Systems (ICEIS 2024) - Volume 1*, pages 61-72
ISBN: 978-989-758-692-7; ISSN: 2184-4992

to remedy this situation but of the 73 used for training, BIRD has only two databases with more than 25 tables and, of the 11 used for development, the largest one has 13 tables.

This paper, therefore, addresses the problem of assessing if LLM-based text-to-SQL tools can be used over relational databases with large schemas, with several hundred objects, counting tables, columns, and foreign keys.

To investigate this problem, the paper abandons the strategy of using many databases with small-to-medium schemas to test LLM-based text-to-SQL tools and concentrates on two challenging databases: a proprietary industrial database, IndDB (the name was 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 (Izquierdo et al., 2021), 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 paper 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 databases are each accompanied by a suite of 100 NL questions that are closely related to those observed in practice.

The paper proceeds to outline nine text-to-SQL strategies, listed in Table 2. The strategies include DANKE, a strategy that combines DANKE with keyword extraction from an NL question (Nascimento et al., 2023), SQLCoder[1] based on an LLM fine-tuned to text-to-SQL, three LangChain-based text-to-SQL strategies[2], "C3 + ChatGPT + Zero-Shot" (Dong et al., 2023), "DIN-SQL + GPT-4" (Pourreza and Rafiei, 2023), and "C3+DIN", a new strategy that combines components of C3 and DIN-SQL.

Then, the paper analyses how the text-to-SQL strategies perform over the Mondial and the IndDB databases, using the suite of 100 NL questions. Except for DANKE and SQLCoder, the paper tested the strategies with GPT-3.5 Turbo and GPT-4.

The paper concludes with a discussion of the results obtained. Briefly, using Mondial, the top-5 strategies with respect to overall accuracy used GPT-4. C3 had the best overall accuracy of 0.78. SQLCoder had a limited overall accuracy of 0.35. Using IndDb, DANKE had the best overall accuracy, closely followed by "Manual prompt chain + DANKE" with GPT-4. All other text-to-SQL strategies had a very low overall accuracy. In general, the limited performance of text-to-SQL strategies for IndDB and Mondial, with large schemas, typical of real-world industrial databases, calls for different text-to-SQL approaches, as discussed in the conclusions.

The paper is organized as follows. Section 2 covers related work. Section 3 describes the IndDB and the Mondial databases. Section 4 summarizes the strategies tested. Section 5 details the experiments. 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) 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[3]; 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

[1] https://huggingface.co/defog/sqlcoder-34b-alpha

[2] https://python.langchain.com/docs/use_cases/qa_structured/sql

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

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.

BIRD – BIg Bench for LaRge-scale Database Grounded Text-to-SQL Evaluation (Li et al., 2023) is a large-scale cross-domain text-to-SQL benchmark in English. The dataset contains 12,751 text-to-SQL data pairs and 95 databases with a total size of 33.4 GB across 37 domains. BIRD 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 (Zhong et al., 2017) has 80,654 NL sentences and SQL annotations of 24,241 tables. Each query in WikiSQL is limited to the same table and does not contain complex operations such as sorting and grouping.

SQL-Eval (Ping, 2023) is a framework[4] that evaluates the correctness of text-to-SQL strategies, created during the development of SQLCoder.

Finally, the sql-create-context[5] 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[6] 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[7] 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 (Gao et al., 2023); "DPG-SQL + GPT-4 + Self-Correction" (no reference available at the time of writing); "DIN-SQL + GPT-4" (Pourreza and Rafiei, 2023); "Hindsight Chain of Thought with GPT-4" (no reference available at the time of writing); and "C3 + ChatGPT + Zero-Shot" (Dong et al., 2023).

The BIRD Web site[8] also publishes a leaderboard with the best-performing tools. At the time of 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 paper were designed.

Other text-to-SQL tools include the Defog SQL-Coder, outlined in Section 4.2. The "60 Top AI Text To SQL Bot Tools" Web site[9] lists other AI tools for text-to-SQL.

Finally, the DB-GPT-Hub[10] 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 DATASETS ADOPTED

A *benchmark dataset* for the text-to-SQL task is a pair $B = (D, \{(L_i, G_i)/i = 1, ..., n\})$, where D is a database and, for $i = 1, ..., n$, L_i is an NL question over D and G_i is the *ground truth* SQL query over D that translates L_i. In the context of this paper, a benchmark dataset is meant exclusively for testing text-to-SQL tools; it is not designed for training such tools. Section 5.2 describes the procedure adopted to evaluate text-to-SQL tools.

The primary benchmark dataset adopted in this paper is IndDB, 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.

[4] Available at https://github.com/defog-ai/sql-eval

[5] https://huggingface.co/datasets/b-mc2/sql-create-context

[6] https://python.langchain.com

[7] https://yale-lily.github.io/spider

[8] https://bird-bench.github.io

[9] https://topai.tools/s/Text-to-SQL-bot

[10] https://github.com/eosphoros-ai/DB-GPT-Hub

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 paper also considers a version of Mondial[11], 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[12] has 46 tables, with a total of 184 columns and 49 foreign keys, some of which are multi-column.

Finally, Table 1 shows basic statistics of the sets of queries, where "#cols" refers to the number of columns of the target clause and the other columns refer to the number of joins, filters, and aggregations that occur anywhere in the query, including any nested query. 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.

4 STRATEGIES TESTED

This section outlines the strategies tested, summarized in Table 2.

4.1 DANKE-Based Strategies

DANKE (Izquierdo et al., 2021) is an automatic, schema-based tool that supports keyword query processing for both the relational and RDF environments (Izquierdo et al., 2018). 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 (García et al., 2017). 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 2 passes the NL query to DANKE.

The "Manual prompt chain + DANKE" strategy (Nascimento et al., 2023) 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[13] is a specialized text-to-SQL model, open-sourced under the Apache-2 license. The latest model, sqlcoder-34b-alpha, 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.

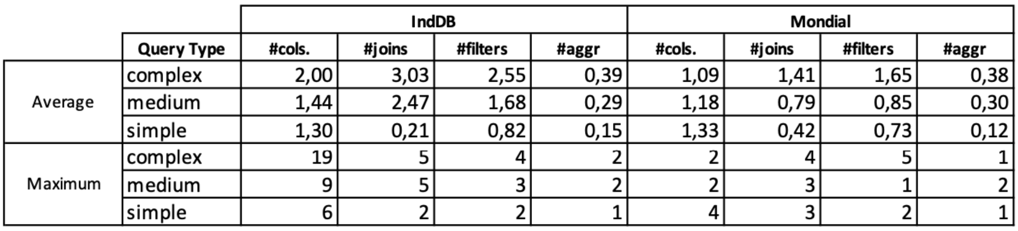
The accuracy of SQLCoder on the Defog's dataset is promising: defog-sqlcoder-34b achieved 84.0%, whereas gpt4-turbo-2023-11-09 achieved 82.5%.

[11] https://www.dbis.informatik.uni-goettingen.de/Mondial/

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

[13] https://huggingface.co/defog/sqlcoder-34b-alpha

Table 1: Basic statistics of the sets of queries.



		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

Table 2: Summary of the strategies tested.

1) **"DANKE"** – The database keyword search tool that is the current interface for IndDB.
2) **"Manual prompt chain + DANKE"** – 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) **"SQLCoder"** – Defog's tool for text-to-SQL, using a special-purpose LLM.
4) **"SQLQueryChain (LangChain)"** – The LangChain SQLQueryChain processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
5) **"SQLDatabaseSequentialChain (LangChain)"** – The LangChain SQLDatabaseSequentialChain processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
6) **"SQL Database Agent (LangChain)"** – The LangChain SQL Database Agent processes NL questions into SQL; tested with GPT-3.5-turbo-16k and GPT-4.
7) **"DIN-SQL"** – An implementation of "DIN-SQL" that allows the use of different LLMs; tested with GPT-3.5-turbo-16k and GPT-4.
8) **"C3"** – An implementation of "C3" that allows the use of different LLMs; tested with GPT-3.5-turbo and GPT-4.
9) **"C3+DIN"** – The C3+DIN tool introduced in Section 4.4; tested with GPT-3.5-turbo-16k and GPT-4.

4.3 LangChain-Based Strategies

LangChain[14] is a framework that helps develop LLM applications. LangChain offers three predefined chains[15] for text-to-SQL, and is compatible with MySQL, PostgreSQL, Oracle SQL, Databricks, SQLite, and other DBMSs.

Very briefly, *SQLQueryChain* extracts metadata from the database automatically, creates a prompt, and passes this metadata to the LLM. This chain greatly simplifies creating prompts to access databases. *SQLDatabaseSequentialChain*, based on the query, determines which tables to use and, based on those tables, calls the SQLQueryChain. This helps when the number of tables in the database is large. Finally, *SQLAgent* provides a more flexible way of interacting with databases. It answers NL questions based on the database schema, as well as on the database content, and recovers from errors by running a generated SQL query, catching the traceback and regenerating it correctly.

In addition to passing the schema in the prompt, these chains for SQL make 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 all LangChain-based strategies with GPT-3.5-turbo-16k and GPT-4.

[14] https://docs.langchain.com

[15] https://python.langchain.com

4.4 C3 and DIN-Based Strategies

4.4.1 C3

"C3 + ChatGPT + Zero-Shot" (Dong et al., 2023) (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).

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 "de-bias" 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" (Pourreza and Rafiei, 2023) (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 (Wei et al., 2023), 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 (Yu et al., 2023).

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

5 EXPERIMENTS

5.1 Configurations

The experiments tested the strategies summarized in Table 2 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.

Additional experiments, not reported here, used other LLMs. The "Manual prompt chain + DANKE" strategy was run with GPT-3.5-turbo-1106, announced on November 6th, 2023, whereas the strategies described in Section 4.3 and 4.4 were run with LLaMA-2-chat, with 70B parameters. However, none of these experiments resulted in good accuracy.

5.2 Evaluation Procedure

Let $B = (D, \{(L_i, G_i)/i = 1, ..., n\})$ be a benchmark dataset. 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 1, 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 1, 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 4 and 5, at the end of paper, show the results for the Mondial and the IndDB databases. Columns under “**Accuracy**” indicate the accuracy results for the simple, medium and complex queries, as well as the overall accuracy; columns “**Input Tokens**” and “**Output Tokens**” respectively show the number of tokens passed as input and received as output from the model; column “**Estim. Cost**” indicates the estimated cost in US Dollars; and column “**Exec. Time**” displays the total time to compute the 100 queries, which naturally depends on the HW and SW setup, and should be used only to compare the strategies.

5.3.1 Results for Mondial

Accuracy. The top-5 strategies with respect to overall accuracy used GPT-4. C3 had the best overall accuracy of 0.78. Then, SQLQueryChain with samples, DIN, and C3+DIN had the same overall accuracy of 0.70. Lastly, SQLQueryChain, without samples, achieved 0.69. Among the Langchain-based strategies, those that passed the entire schema and samples in the prompt had superior overall accuracy. 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. SQLDatabaseSequentialChain and SQLAgent had minimal cost due to the smaller prompts obtained by filtering the schemas for the relevant tables. However, GPT-3.5-turbo and GPT-4 misidentified crucial tables, leading to incorrect SQL queries. SQLAgent became lost or hallucinated using GPT-4. In fact, SQLAgent is not fully compatible with GPT-4. Also, SQLAgent had a poor performance with GPT-3.5-turbo.

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

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

Table 3 shows the error analysis of C3 with GPT-4. When compared with that of C3 for the Spider benchmark (Dong et al., 2023), it indicates that the errors resulting from schema linking and joins are exacerbated in the experiments with Mondial, which would be expected, given that the Mondial schema is far more complex than the majority of the datasets in the Spider benchmark.

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

5.3.2 Results for IndDB

Accuracy. “DANKE” had the best overall accuracy, followed by “Manual prompt chain + DANKE” with GPT-4, but both were low. All other strategies achieved a very low overall accuracy. Of these strategies, C3+DIN with GPT-4 had the best overall accu-

Table 3: Error analysis of the C3 with GPT-4 experiment using the Mondial database.

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

racy, 0.13, but at a high cost, and SQLQueryChain with GPT-4 had the second-best overall accuracy, 0.12, but with a much lower cost.

Strategy Details. SQLQueryChain with GPT-4 and using samples could not be run since it required more tokens than the allowed 8,192 for GPT-4. 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. SQLCoder was not tested over IndDB for privacy reasons.

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 22 strategies (the number of lines of Table 5), the analysis concentrated on LangChain SQLQueryChain with GPT-4, the second-best accuracy of the text-to-SQL strategies with a low cost.

The analysis revealed some problems with the experiments with IndDB: (1) some of the NL questions in the ground truth were incomplete or ambiguous; (2) certain ground truth SQL queries were very specific translations of the NL question, either because they had columns in the target clause that were specific interpretations of terms in the NL question, or because the where clause contained conditions that were again specific interpretations of filters in the NL question; (3) 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); (4) the relational schema is relatively large, which led to LLM hallucination; (5) many NL questions led to SQL queries with up to 5 joins to gather all the required data.

The ground truth was then adjusted to minimize Problems (1) and (2), and the experiment with LangChain SQLQueryChain with GPT-4 was re-executed. The accuracies thereby obtained were indeed much better than those reported in Table 5 for this strategy. For example, the accuracy of simple NL questions improved from 0.27 to 0.48, which reflects the adjustments of the ground truth.

However, an accuracy of 0.48 for simple NL questions over IndDB is still much lower than the accuracy of 0.82 that this strategy obtained for simple NL questions over Mondial, reported in Table 4. Since the Mondial relational schema uses familiar table and column names, this observation suggests that problem (3) may be blamed for the lower accuracy of IndDB.

The traditional approach to avoid problem (3) is to create views over the relational schema and carefully select view and column names that better match user terms. However, this approach faced restrictions to be implemented on IndDB, which is a production database, and it could not be tested in the experiments.

Problems (4) and (5) were expected since the experiments aimed at stressing the text-to-SQL strategies with a large schema and complex questions.

6 CONCLUSIONS

The experiments with Mondial indicated that C3 with GPT-4 had the best overall accuracy. However, C3 incurred higher costs, longer runtime, and more output tokens. SQLQueryChain with GPT-4 proved more effective, concerning token cost and execution time. In general, passing the entire schema to the LLM achieved better results than filtering the schema and passing just a few tables, but this approach is limited by the number of tokens the LLM allows, especially for complex schemas, such as that of Mondial.

Regarding IndDB, DANKE had the best overall accuracy, closely followed by “Manual prompt chain + DANKE” with GPT-4, albeit both results were low. This suggests that the NL questions defined for IndDB were quite challenging. All other strategies had an overall accuracy of less than 15%, which is very low. They correctly processed only those NL questions over IndDB that mapped to a single table SQL queries.

C3+DIN did not have good performance. 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.

The specific goal of this paper, to use an LLM-based text-to-SQL tool to construct a Natural Language Interface for IndDB as an alternative to DANKE, therefore, failed for the list of text-to-SQL strategies tested, even though this list includes some strategies that ranked high in familiar leaderboards, and a model fine-tuned for the text-to-SQL task, with a reported very good performance. In general, the poor performance of text-to-SQL strategies for the large, complex databases typical of real-world industrial applications calls for different alternatives and this is the take-home lesson of this paper.

Future work will consider three alternatives for text-to-SQL, which are not mutually exclusive.

The first alternative will explore a combination of a question-and-answer interface and database views. As hinted at the end of Section 5, the view definitions will: (1) create a vocabulary that better matches user terms; (2) predefine frequently required joins. A user session will start with a step-by-step NL question specification to cope with any mismatch between the user terms and the view vocabulary, and to disambiguate term usage. After this step, the interface will select a few views to apply a text-to-SQL strategy.

The second alternative will consider fine-tuning a locally stored LLM specifically for a given database with a large schema. The training dataset can be quite laborious to create, but GPT-4 may come in hand to augment the training set from a seed set of NL questions and their SQL translations.

Suppose that a reasonable set P of pairs (S', Q'), consisting of an NL question S' and its SQL translation Q', can indeed be generated for a given database D. A third alternative would be to sample P to construct a set E of pairs (S', Q') such that S and S' are similar, for a given NL question S. Then, a text-to-SQL strategy would pass E to the LLM to help translate S, as already used in a limited form in the LangChain strategies with samples.

ACKNOWLEDGEMENTS

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

REFERENCES

Affolter, K., Stockinger, K., and Bernstein, A. (2019). A comparative survey of recent natural language interfaces for databases. *The VLDB Journal*, 28:793–819.

Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., lu Chen, Lin, J., and Lou, D. (2023). C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B., and Zhou, J. (2023). Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

García, G. M., Izquierdo, Y. T., Menendez, E., Dartayre, F., and Casanova, M. A. (2017). Rdf keyword-based query technology meets a real-world dataset. In *Proceedings of the International Conference on Extending Database Technology*, pages 656–667. OpenProceedings.org.

Izquierdo, Y. T., Garcia, G. M., Lemos, M., Novello, A., Novelli, B., Damasceno, C., Leme, L. A. P. P., and Casanova, M. A. (2021). A platform for keyword search and its application for covid-19 pandemic data. *Journal of Information and Data Management*, 12(5):656–667.

Izquierdo, Y. T., García, G. M., Menendez, E. S., Casanova, M. A., Dartayre, F., and Levy, C. H. (2018). Quiow: a keyword-based query processing tool for rdf datasets and relational databases. In *International Conference on Database and Expert Systems Applications (DEXA)*, pages 259–269. Springer.

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.

Li, J. et al. (2023). Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111*.

Nascimento, E. R., Garcia, G. M., Victorio, W. Z., Lemos, M., Izquierdo, Y. T., Garcia, R. L., Leme, L. A. P., and Casanova, M. A. (2023). 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*, volume 3618 of *CEUR Workshop Proceedings*.

Ping, W. J. (2023). Open-sourcing sqleval: our framework for evaluating llm-generated sql.

Pourreza, M. and Rafiei, D. (2023). Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023). Chain-of-thought prompting elicits reasoning in large language models. *arXiv preprint arXiv:2201.11903*.

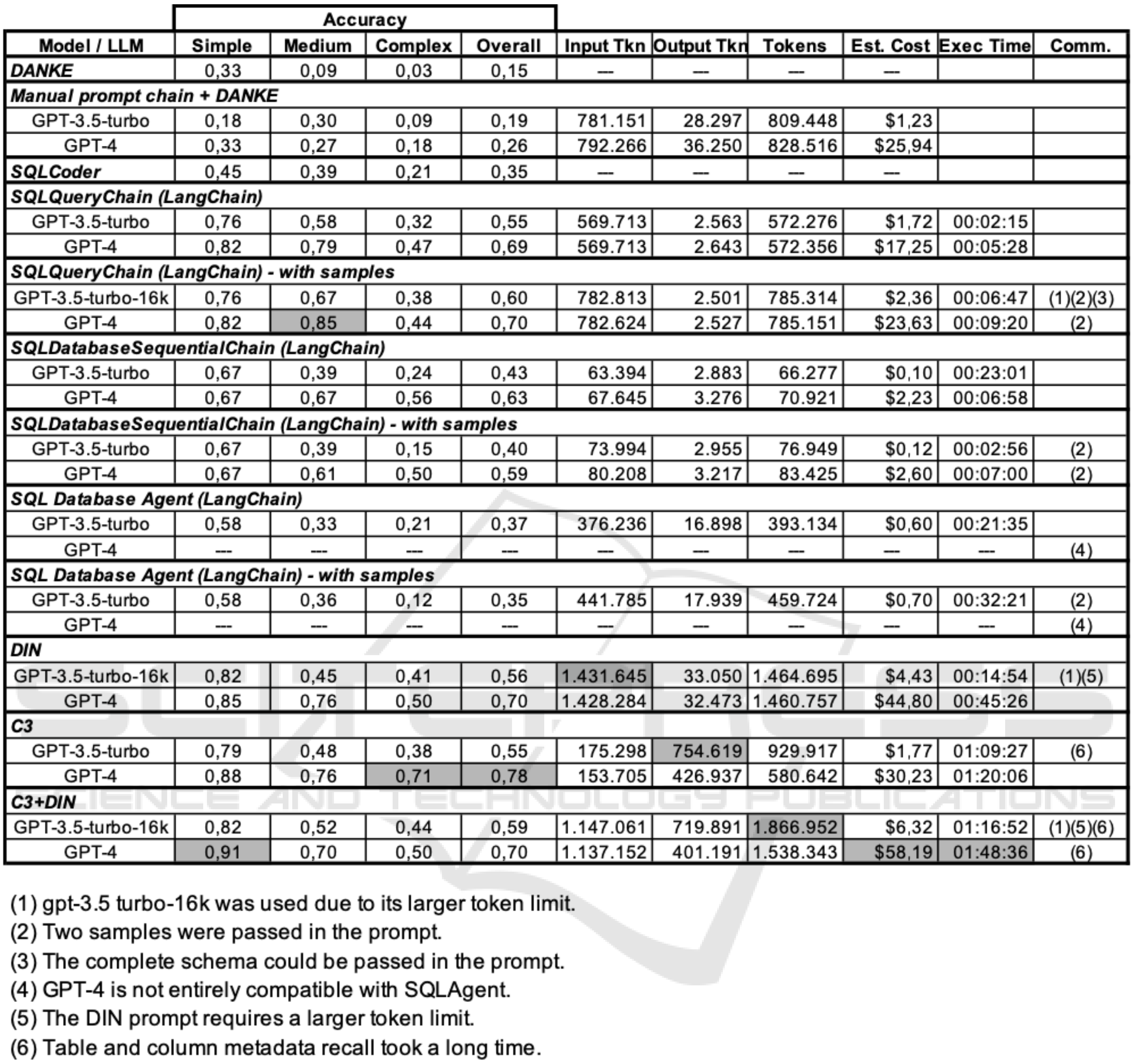
Yu, T. et al. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium.

Yu, X., Cheng, H., Liu, X., Roth, D., and Gao, J. (2023). Automatic hallucination assessment for aligned large language models via transferable adversarial attacks. *arXiv preprint arXiv:2310.12516*.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

APPENDIX

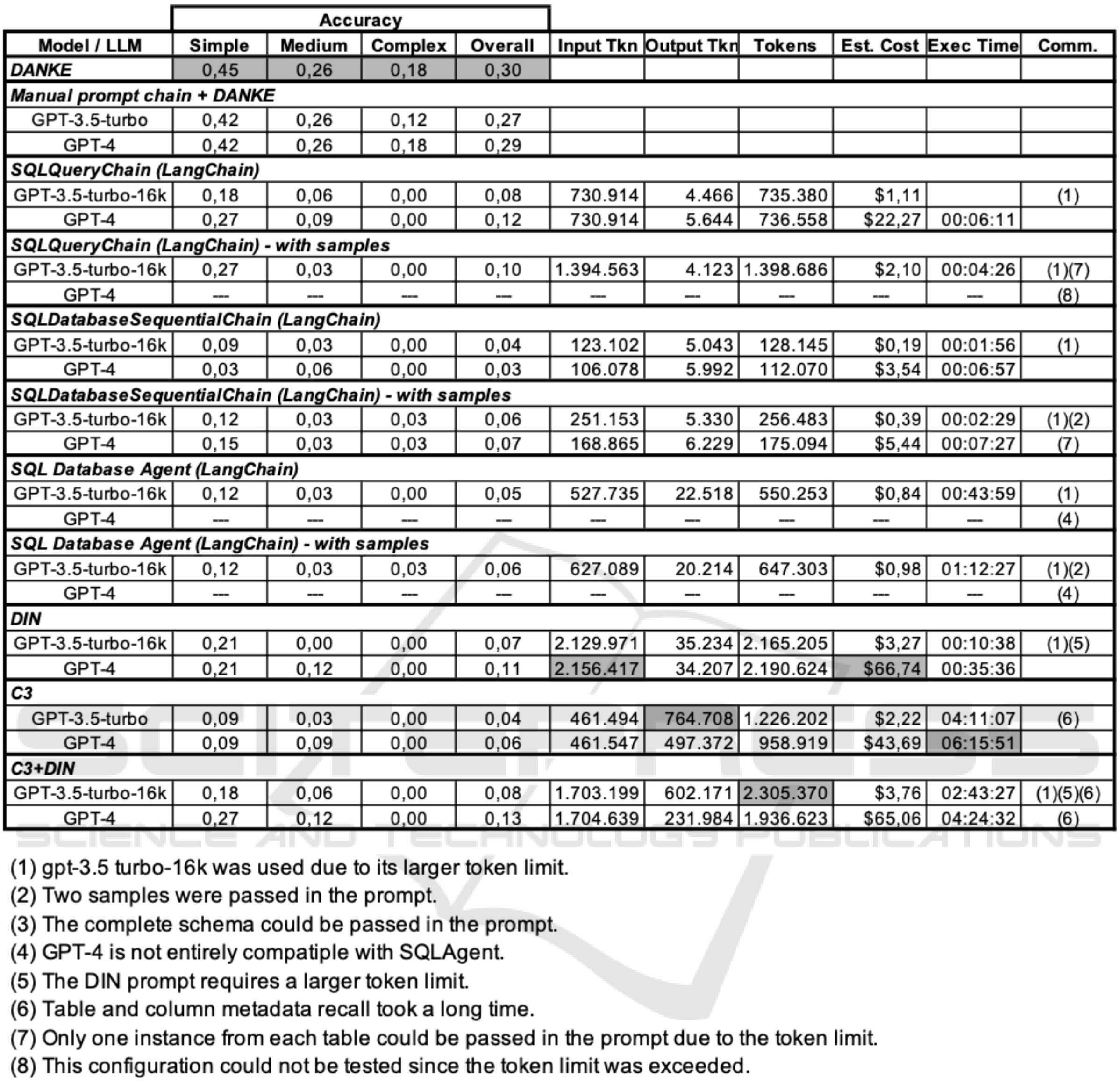
Table 4: Results for Mondial.



	Accuracy									
Model / LLM	Simple	Medium	Complex	Overall	Input Tkn	Output Tkn	Tokens	Est. Cost	Exec Time	Comm.
DANKE	0,33	0,09	0,03	0,15	—	—	—	—		
Manual prompt chain + DANKE										
GPT-3.5-turbo	0,18	0,30	0,09	0,19	781.151	28.297	809.448	$1,23		
GPT-4	0,33	0,27	0,18	0,26	792.266	36.250	828.516	$25,94		
SQLCoder	0,45	0,39	0,21	0,35	—	—	—	—		
SQLQueryChain (LangChain)										
GPT-3.5-turbo	0,76	0,58	0,32	0,55	569.713	2.563	572.276	$1,72	00:02:15	
GPT-4	0,82	0,79	0,47	0,69	569.713	2.643	572.356	$17,25	00:05:28	
SQLQueryChain (LangChain) - with samples										
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)
GPT-4	0,82	0,85	0,44	0,70	782.624	2.527	785.151	$23,63	00:09:20	(2)
SQLDatabaseSequentialChain (LangChain)										
GPT-3.5-turbo	0,67	0,39	0,24	0,43	63.394	2.883	66.277	$0,10	00:23:01	
GPT-4	0,67	0,67	0,56	0,63	67.645	3.276	70.921	$2,23	00:06:58	
SQLDatabaseSequentialChain (LangChain) - with samples										
GPT-3.5-turbo	0,67	0,39	0,15	0,40	73.994	2.955	76.949	$0,12	00:02:56	(2)
GPT-4	0,67	0,61	0,50	0,59	80.208	3.217	83.425	$2,60	00:07:00	(2)
SQL Database Agent (LangChain)										
GPT-3.5-turbo	0,58	0,33	0,21	0,37	376.236	16.898	393.134	$0,60	00:21:35	
GPT-4	---	---	---	---	—	—	—	—	---	(4)
SQL Database Agent (LangChain) - with samples										
GPT-3.5-turbo	0,58	0,36	0,12	0,35	441.785	17.939	459.724	$0,70	00:32:21	(2)
GPT-4	---	---	---	---	—	—	—	—	---	(4)
DIN										
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)(5)
GPT-4	0,85	0,76	0,50	0,70	1.428.284	32.473	1.460.757	$44,80	00:45:26	
C3										
GPT-3.5-turbo	0,79	0,48	0,38	0,55	175.298	754.619	929.917	$1,77	01:09:27	(6)
GPT-4	0,88	0,76	0,71	0,78	153.705	426.937	580.642	$30,23	01:20:06	
C3+DIN										
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)
GPT-4	0,91	0,70	0,50	0,70	1.137.152	401.191	1.538.343	$58,19	01:48:36	(6)

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

Table 5: Results for IndDB.



	Accuracy									
Model / LLM	**Simple**	**Medium**	**Complex**	**Overall**	**Input Tkn**	**Output Tkn**	**Tokens**	**Est. Cost**	**Exec Time**	**Comm.**
DANKE	0,45	0,26	0,18	0,30						
Manual prompt chain + DANKE										
GPT-3.5-turbo	0,42	0,26	0,12	0,27						
GPT-4	0,42	0,26	0,18	0,29						
SQLQueryChain (LangChain)										
GPT-3.5-turbo-16k	0,18	0,06	0,00	0,08	730.914	4.466	735.380	$1,11		(1)
GPT-4	0,27	0,09	0,00	0,12	730.914	5.644	736.558	$22,27	00:06:11	
SQLQueryChain (LangChain) - with samples										
GPT-3.5-turbo-16k	0,27	0,03	0,00	0,10	1.394.563	4.123	1.398.686	$2,10	00:04:26	(1)(7)
GPT-4	---	---	---	---	---	---	---	---	---	(8)
SQLDatabaseSequentialChain (LangChain)										
GPT-3.5-turbo-16k	0,09	0,03	0,00	0,04	123.102	5.043	128.145	$0,19	00:01:56	(1)
GPT-4	0,03	0,06	0,00	0,03	106.078	5.992	112.070	$3,54	00:06:57	
SQLDatabaseSequentialChain (LangChain) - with samples										
GPT-3.5-turbo-16k	0,12	0,03	0,03	0,06	251.153	5.330	256.483	$0,39	00:02:29	(1)(2)
GPT-4	0,15	0,03	0,03	0,07	168.865	6.229	175.094	$5,44	00:07:27	(7)
SQL Database Agent (LangChain)										
GPT-3.5-turbo-16k	0,12	0,03	0,00	0,05	527.735	22.518	550.253	$0,84	00:43:59	(1)
GPT-4	---	---	---	---	---	---	---	---	---	(4)
SQL Database Agent (LangChain) - with samples										
GPT-3.5-turbo-16k	0,12	0,03	0,03	0,06	627.089	20.214	647.303	$0,98	01:12:27	(1)(2)
GPT-4	---	---	---	---	---	---	---	---	---	(4)
DIN										
GPT-3.5-turbo-16k	0,21	0,00	0,00	0,07	2.129.971	35.234	2.165.205	$3,27	00:10:38	(1)(5)
GPT-4	0,21	0,12	0,00	0,11	2.156.417	34.207	2.190.624	$66,74	00:35:36	
C3										
GPT-3.5-turbo	0,09	0,03	0,00	0,04	461.494	764.708	1.226.202	$2,22	04:11:07	(6)
GPT-4	0,09	0,09	0,00	0,06	461.547	497.372	958.919	$43,69	06:15:51	
C3+DIN										
GPT-3.5-turbo-16k	0,18	0,06	0,00	0,08	1.703.199	602.171	2.305.370	$3,76	02:43:27	(1)(5)(6)
GPT-4	0,27	0,12	0,00	0,13	1.704.639	231.984	1.936.623	$65,06	04:24:32	(6)

(1) gpt-3.5 turbo-16k was used due to its larger token limit.
(2) Two samples were passed in the prompt.
(3) The complete schema could be passed in the prompt.
(4) GPT-4 is not entirely compatiple with SQLAgent.
(5) The DIN prompt requires a larger token limit.
(6) Table and column metadata recall took a long time.
(7) Only one instance from each table could be passed in the prompt due to the token limit.
(8) This configuration could not be tested since the token limit was exceeded.