Text-to-SQL Experiments with Engineering Data Extracted from CAD Files

Júlio G. Campos[1][a], Grettel M. García[1][b], Jefferson A. de Sousa[1][c], Eduardo T. L. Corseuil[1][d], Yenier T. Izquierdo[1][e], Melissa Lemos[1,2][f] and Marco A. Casanova[1,2][g]

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

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

Keywords: Text-to-SQL, Large Language Models, Relational Databases, Llama, Gemma, GPT.

Abstract: The development of Natural Language (NL) interfaces to access relational databases attracted renewed interest with the use of Large Language Models (LLMs) to translate NL questions to SQL queries. This translation task is often referred to as text-to-SQL, a problem far from being solved for real-world databases. This paper addresses the text-to-SQL task for a specific type of real-world relational database storing data extracted from engineering CAD files. The paper introduces a prompt strategy tuned to the text-to-SQL task over such databases and presents a performance analysis of LLMs of different sizes. The experiments indicated that GPT-4o achieved the highest accuracy (96%), followed by Llama 3.1 70B Instruct (86%). Quantized versions of Gemma 2 27B and Llama 3.1 8B had a very limited performance. The main challenges faced in the text-to-SQL task involved SQL complexity and balancing speed and accuracy when using quantized open-source models.

1 INTRODUCTION

The development of Natural Language (NL) interfaces to access relational databases attracted renewed interest with the use of Large Language Models (LLMs) to translate NL questions to SQL queries. This translation task is often referred to as *text-to-SQL*.

Companies are increasingly interested in using text-to-SQL to support business processes that require easy access to databases at both the operational and strategic levels. However, they need to address several challenges to make this technology viable, such as semantic data integration, which allows LLMs not only to understand how different databases relate to each other but also to understand the semantics of the data (Campos et al., 2023).

Concerns about cost and data privacy are also driving companies to seek alternatives to proprietary LLMs. Despite their constant evolution, the potential of open-source LLMs is under-exploited, even as notable advances have been made in programming tasks, mathematical reasoning, and text generation (Chiang et al., 2023). Much of the research on text-to-SQL focuses primarily on proprietary LLMs, leaving open-source LLMs behind (Gao et al., 2023).

This paper addresses the text-to-SQL task for one very specific type of real-world relational database storing data extracted from engineering CAD files. The paper introduces a prompt strategy tuned to the text-to-SQL task over such databases and presents a performance analysis of LLMs of different sizes, both proprietary and open source.

The paper assumes that the objects and their property values are first extracted from engineering Computer-Aided Design (CAD) files, which represents 3D models of industrial plants. Then, the data extracted are stored in a relational table with three columns, called an *object-property-value* table. Each row in the table has the format (o, p, v), where o is an object ID, p is the name of one of the object properties, and v is the property value. The use of triples is a familiar strategy to address the enormous variability of the object properties.

[a] https://orcid.org/0000-0002-5023-3836
[b] https://orcid.org/0000-0001-9713-300X
[c] https://orcid.org/0000-0002-5928-9959
[d] https://orcid.org/0000-0002-7543-4140
[e] https://orcid.org/0000-0003-0971-8572
[f] https://orcid.org/0000-0003-1723-9897
[g] https://orcid.org/0000-0003-0765-9636

Campos, J. G., García, G. M., A. de Sousa, J., Corseuil, E. T. L., Izquierdo, Y. T., Lemos, M. and Casanova, M. A.
Text-to-SQL Experiments with Engineering Data Extracted from CAD Files.
DOI: 10.5220/0013436800003929
In *Proceedings of the 27th International Conference on Enterprise Information Systems (ICEIS 2025) - Volume 1*, pages 343-350
ISBN: 978-989-758-749-8; ISSN: 2184-4992

Albeit it would seem more natural to store such triples in RDF (Resource Description Framework)[1], there are two basic reasons for opting for a relational approach. First, on the practical side, relational database systems are still the preferred option of IT teams, a tradition that is difficult to escape. Second, on the data modeling side, the value v extracted from the CAD files is frequently a string of the form "`5 mm`" (or a much more complex string), that is, v represents the property value and the unit of measure. This suggests that it would be preferred to use quadruples (o, p, v, u), separating the unit u from the value v, which is still not difficult to model as a relational table with four columns. By contrast, in RDF, one would have to reify the value and use three triples, say, (o, p, i), $(i, : value, v)$, and $(i, : unit, u)$, which complicates synthesizing SPARQL queries.

However, storing data as triples poses two challenges for the text-to-SQL task. The first challenge lies in that the second column of the table stores the property names, which must be linked to the terms used in NL questions; in the familiar text-to-SQL task, this *schema linking step* is much simpler since it is performed mostly against the table or column names that occur in the relational schema. The second challenge is that retrieving objects based on multiple filters over distinct property values requires multiple joins of the *object-property-value* table with itself; in the familiar text-to-SQL task, the *SQL query compilation step* does not require such multiple joins, since a single tuple would store all (or most) property values of an object.

The first contribution of this paper is a prompt strategy to help an LLM in the text-to-SQL task in this specific scenario. The prompt strategy has instructions for constructing SQL queries over an *object_property_value* table, and examples of SQL templates covering frequent NL questions the users submit.

The second contribution of the paper is an evaluation, using real data and the proposed prompt strategy, of four LLMs, where three are open-source – LLama 3.1 70B Instruct, Gemma 2 27B, and Llama 3.1 8B – and one is proprietary – GPT-4o. The experiments indicated that GPT-4o achieved the highest accuracy (96%), followed by Llama 3.1 70B Instruct (86%). Quantized versions of Gemma 2 27B and Llama 3.1 8B had a low performance. The results suggested that the proposed prompt strategy was essential to achieve proper performance, and that balancing speed and accuracy, when using quantized open-source models, remained challenging.

The paper is structured as follows. Section 2 presents related work. Section 3 introduces the database modeling adopted in the paper. Section 4 details the proposed prompt strategy. Section 5 describes the experiments. Section 6 discusses the results obtained in the experiments. Finally, Section 7 contains the final remarks and suggests future work.

[1]https://www.w3.org/TR/rdf11-concepts/

2 RELATED WORK

Text-to-SQL is an established task in the field of natural language processing (NLP), aimed at translating questions in natural language into SQL queries, thereby facilitating access to databases for users without specialized technical knowledge.

Before the popularization of large language models (LLMs), two main streams addressed the text-to-SQL task. The first was based on using sequence-to-sequence (Seq2Seq) models (Sutskever et al., 2014). An encoder would analyze the schema structure of related tables and understand the semantics of the natural language question. Then, a decoder would generate the tokens of the SQL query, producing them one by one based on the encoded representation of the question. The second paradigm involved fine-tuning pre-trained language models (PLMs), such as BERT (Devlin et al., 2019), which leverage the knowledge acquired from large collections of texts. These models demonstrated significant effectiveness in enhancing the performance of text-to-SQL mapping tasks in later contexts (Shi et al., 2024).

Regarding the state-of-the-art (SOTA), the current literature on text-to-SQL based on LLMs focuses on two main approaches: the first is *prompt engineering*, which does not require retraining the model and takes advantage of LLMs' ability to be directed toward specific tasks through the addition of data, instructions, and examples to the prompt. This approach offers greater flexibility by allowing the use of simple techniques, such as few-shot learning (Brown et al., 2020) and Chain-of-Thought (CoT) (Wei et al., 2024), as well as more sophisticated techniques based on Retrieval-Augmented Generation (RAG) (Lewis et al., 2020), which enhance large language models (LLMs) by combining them with external knowledge bases. The second is *fine-tuning*, which involves training an LLM on text-to-SQL-specific datasets. Comparing the two approaches, prompt engineering requires fewer data, is faster, and can yield good results. However, fine-tuning tends to improve LLM performance, but requires a more extensive training dataset (Shi et al., 2024).

With the use of LLMs, there has been a significant increase in the accuracy of text-to-SQL experiments

on benchmarks, such as Spider (Yu et al., 2019), continuously raising the state of the art in this field. To give an idea of the progress, in 2023, the execution accuracy on Spider increased from around 73% to 91.2%. A significant limitation of Spider, though, is that its dataset primarily comprises simple SQL queries that use few tables, which does not reflect the complexity of most real-world databases. Therefore, more realistic benchmarks have been created for testing LLMs, such as BIRD (Li et al., 2024), Dr. Spider (Chang et al., 2023), and Spider 2.0 (Lei et al., 2024).

Spider 2.0[2], an evolution of Spider 1.0, is a framework comprising 632 real-world text-to-SQL workflow problems, derived from enterprise-level database use cases. The databases in Spider 2.0 are sourced from real data applications, often containing over 1,000 columns and stored in local or cloud database systems such as BigQuery and Snowflake. Solving problems in Spider 2.0 frequently requires understanding and searching through database metadata, dialect documentation, and even project-level codebases. Early evaluations indicate that, based on OpenAI o1-preview[3], the code agent framework successfully solves only 17.0% of the tasks, compared with 91.2% on Spider 1.0 and 73.0% on BIRD. The results on Spider 2.0 show that while language models demonstrated remarkable performance in code generation – especially in prior text-to-SQL benchmarks – they require significant improvement to achieve adequate performance for real-world enterprise usage.

A comprehensive survey of text-to-SQL strategies can be found in (Shi et al., 2024), including a discussion on benchmark datasets, prompt engineering, and fine-tuning methods. The Web site Awesome Text2SQL[4] lists tools with the best performance across various text-to-SQL benchmarks, and the DB-GPT-Hub project[5] explores how to use LLMs for text-to-SQL.

In particular, LangChain[6] is a framework that helps develop text-to-SQL strategies using LLMs. LangChain is compatible with MySQL, PostgreSQL, Oracle SQL, Databricks, SQLite, and other DBMSs. Very briefly, the *LangChain SQLQueryChain* automatically extracts database metadata, creates a prompt, and passes this metadata to the model. This chain greatly simplifies the creation of prompts to access databases. In addition to including the schema in the prompt, this chain allows providing sample data that can help a model build correct queries when the data format is unclear. Sample rows are added to the prompt after the column information for each corresponding table.

Some recent work on the text-to-SQL task based on LLMs has obtained promising results using real-world data. (Coelho et al., 2024) proposes a technique based on Retrieval-Augmented Generation (RAG) (Lewis et al., 2020) to enhance the performance of LLMs in the text-to-SQL task on a real-world relational database from an energy company.

Finally, this paper differs from the (relational) text-to-SQL approaches in that it focuses on a very specific relational database storing triples. It also differs from text-to-SPARQL strategies, such as that described in (Avila et al., 2024), developed for generic RDF datasets.

3 THE PROPOSED DATABASE CONSTRUCTION PROCESS

This section describes the database construction process based on data extracted from engineering Computer-Aided Design (CAD) files, which represents models of industrial plants. The examples that follow are based on data from a floating production storage and offloading (FPSO) unit of an oil and gas company, used in Section 5 for the experiments, but the process is general and typical of engineering data.

The database construction process has the following basic steps. First, the raw data are extracted from the JSON (JavaScript Object Notation) files of engineering projects stored in a Computer-Aided Design (CAD) information system, which represents models of industrial plants. The data refer to hundreds of properties related to materials, weight, temperature, pressure, location, among others, with their corresponding values. The data must undergo a cleaning process to ensure completeness and consistency. Then, the resulting cleaned data is also stored as JSON files.

Next, the JSON files are remapped and stored in a relational table, called an *object-property-value* table, with three columns: *id*, *property*, and *value* (see Table 1). The *id* column stores object IDs, the *property* column stores property names, and the *value* column stores the values of these properties.

After ingesting the data into the *object-property-value* table, the table rows are analyzed to investigate their nature. The *value* column stores heterogeneous data and may include *null* values. Also, several numeric values appear along with their units of mea-

[2]The code, baseline models, and data are available at https://spider2-sql.github.io

[3]https://openai.com/index/introducing-openai-o1-preview/

[4]https://github.com/eosphoros-ai/Awesome-Text2SQL

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

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

Table 1: Snippet from *object-property-value* table.

id	property	value
ID-1	Length	10 mm
ID-2	Max Temperature	25 C
ID-2	Weight	100 kg
ID-3	Material	Carbon Steels
ID-3	Created on	01/01/2000 10:25:35
...	...	...

sure, typically for properties such as weight, height, elevation, pressure, and so on. This presents a significant challenge for LLMs which need to generate SQL queries that can handle these idiosyncrasies.

As mentioned in the introduction, a value extracted from the CAD files is frequently a string of the form "`5 mm`" (or a much more complex string), that is, the value represents the property value and the unit of measure. This suggests that it would be preferred to use quadruples (o, p, v, u), separating the unit u from the value v, which is still not difficult to model as a relational table with four columns. The implementation used in the experiments described in Section 5 does not take into account this further transformation since it proved difficult to separate the property value from the unit of measure in the real-world data used. This point thus requires further investigation.

4 THE PROPOSED PROMPT STRATEGY

The proposed prompt strategy to help an LLM in the text-to-SQL task for an *object-property-value* table, as described in Section 3, is divided into three main blocks – context, instructions, and examples – as shown below.

```
# Context: {context}
# Instructions: {instructions}
# Examples: {examples}

Create an SQL query based on the best
example that matches the question below:
```

In the context block, the LLM is conditioned to assume the role of a database expert, whose task is to generate a valid SQL query for the database in question. The structure of the table and its columns is prompted to the LLM. A list of all property names that occur in the *property* column is added to this section.

The second block contains specific instructions for constructing an SQL query, such as using the *property* column whenever a question refers to a property, and the *value* column when it relates to the value of that property. In this block, a list is also added with all property names that are categorical and their respective values. These values must always be used when constructing the SQL query.

In the implementation used in Section 5, it should be noted that the questions are asked in Portuguese and all data stored in the database are in English. Therefore, the LLM must be able to identify a property of an object mentioned in the question in Portuguese and identify the English translation from the list provided in the prompt. This posed an additional challenge for the text-to-SQL task.

The LLM is also instructed always to return the SQL query on a single line without any formatting to make it easier to compare it with the expected SQL. Furthermore, the LLM is asked never to explain its answer, but just to return the SQL code.

In the last block, eight generic SQL examples are inserted, one for each type of clause or function frequently observed in the translations of the user questions. These examples serve as templates for the LLM to use as references.

Below the examples, the LLM is instructed to generate the SQL query based on the best corresponding example.

For instance, one of the eight SQL examples inserted in the prompt is:

```
- To select objects that have more than one
property with their specific values:

   SELECT DISTINCT tp1.id
    FROM object-property-value tp1
    JOIN object-property-value tp2
        ON tp1.id = tp2.id
    WHERE tp1.property = 'p1'
        AND tp1.value = 'v1'
        AND tp2.property = 'p2'
        AND tp2.value = 'v2'
```

which indicates how to perform a self-join on the *object-property-value* table, where `p1` and `p2` represent generic properties, and `v1` and `v2` their corresponding values.

5 EXPERIMENTAL SETUP

5.1 LLMs Tested

The experiments tested four LLMs of different sizes (see Table 2): GPT-4o 2024-05-13[7], Llama 3.1 70B Instruct, Llama 3.1 8B[8], and Gemma 2 27B[9]. GPT-

[7]https://openai.com/index/hello-gpt-4o/
[8]https://llama.meta.com/
[9]https://ai.google.dev/gemma/

4o is a proprietary model from OpenAI, and the other three are open-source models from Meta and Google.

5.2 Hardware and Software Setup

GPT-4o was run at OpenAI via the Microsoft Azure cloud computing platform. Llama 3.1 70B Instruct was run on HuggingChat[10]. Llama 3.1 8B and Gemma 2 27B were downloaded using Ollama[11] with a 4-bit quantization level (Q4_0), which reduces their original size to allow execution on computers without powerful hardware. They were run on a personal computer with a 10th-generation Intel Core i7 processor, featuring six processing cores, 32 GB of RAM, and a 64-bit Windows operating system. All models were executed with the temperature setting set to 0 to minimize response variability and achieve more predictable results.

5.3 Benchmark

The benchmark consisted of a database with data from a floating production storage and offloading (FPSO) unit of an oil and gas company, constructed as described in Section 3. The resulting *object-property-value* table contained 312,982 rows, related to 6,228 distinct objects, and 110 distinct properties, among which 37 have categorical values about classes, subclasses, standard categories and types of equipment.

The benchmark featured 50 NL questions and their ground truth SQL translations. The NL questions were formulated in Portuguese, covering different SQL clauses and functions, as summarized in Table 3.

The questions were designed from the perspective of an inexperienced database user who would like to ask questions about one or more properties of one or more engineering equipment. Also, the user might not know which units of measurement are stored in the database for some properties, such as weight, width, pressure, etc. Therefore, some questions were formulated with units of measurement different from those that appear in the database, such as tons instead of kilograms. The benchmark also included questions about equipment that has properties with null values, reflecting the reality of the data.

Each NL question was accompanied by a *ground-truth* SQL translation and the result of its execution on the database. For example, an NL question involving a self-join and its ground-truth SQL translation is (data were anonymized with asterisks to preserve confidentiality):

Question: "*List in alphabetical order all the objects that are 100mm long and have fluid code '*'.*"

SQL:

```
SELECT DISTINCT t1.id
FROM object-property-value t1
JOIN object-property-value t2
    ON t1.id = t2.id
WHERE t1.property =  '*'
    AND t1.value = '100 mm'
    AND t2.property = '*'
    AND t2.value =  '*'
ORDER BY t1.id
```

5.4 Performance Measure

The experiments measured the *accuracy* of the LLM text-to-SQL task over the benchmark, defined as the percentage of NL questions correctly translated, as usual.

Let Q_N be an NL question, Q_{GT} be its ground truth SQL query, and Q_{LLM} be the SQL query the LLM generated for Q_N. The correctness of Q_{LLM} was computed as follows. Q_{LLM} was first syntactically compared with Q_{GT}. If Q_{LLM} and Q_{GT} were not identical, Q_{LLM} was tentatively marked as incorrect and executed on the database for further validation. If the execution of Q_{LLM} produced the expected results for Q_N, Q_{LLM} was re-evaluated as correct. This process eliminated false negatives.

The incorrect SQL queries were analyzed to discover new approaches for improving the prompt. Thus, the incorrect SQL queries provided insights to increase accuracy.

6 RESULTS AND DISCUSSION

Table 4 shows the results obtained in the experiments. The columns of the table correspond to the model, model parameters, duration (elapsed time to run all 50 NL questions), number of correct answers ("Hits"), number of incorrect answers ("Errors"), and the accuracy. GPT-4o achieved the best accuracy of 96%, followed by Llama 3.1 70B, which achieved 86%. Gemma 2 27B reached 28%, and Llama 3.1 8B just 10%.

It should be noted that, without the proposed prompt strategy, all models achieved a negligible performance on the benchmark (results not included in Table 4). This was indeed expected, given the challenges of schema linking and SQL translation already pointed in the introduction.

GPT-4o took around one minute to answer all NL questions, the least time of all experiments. Llama

[10]https://huggingface.co/chat/

[11]https://ollama.com/

Table 2: LLM models used in the experiments.

LLM Model	Params	Context Window	Quantization Level	Owner	Released
Llama 3.1	8B	128K	Q4_0	Meta	07/23/2024
Llama 3.1 Instruct	70B	128K	-	Meta	07/23/2024
Gemma 2	27B	8K	Q4_0	Google	06/27/2024
GPT-4o	-	128K	-	OpenAI	05/13/2024

Table 3: Distribution of the ground truth SQL queries.

Clauses and Aggregate Functions	Questions	Quantity
SELECT[a]	01-10	10
COUNT	11-20	10
SELF-JOIN	21-30	10
AVG	31-40	10
MAX	41-43	3
MIN	44-46	3
GROUP BY and HAVING	47-50	4

[a] Only SELECT projections.

3.1 8B, the smallest of all models, took approximately 21 minutes, while Gemma 2 27B, the medium-sized model, took about 1 hour. However, these results are unfair since these open-source models were run on a (very) small local server. The duration for Llama 3.1 70B Instruct was not measured because questions had to be manually submitted to HuggingChat. However, the response time for each question was relatively quick, taking just a few seconds.

GPT-4o generated just two wrong SQL queries. In the first query, GPT-4o generated a WHERE clause with a carbon steel material value different from that of the ground truth SQL query. A more detailed analysis of the types of materials in the database revealed that "carbon steel" was written as "Carbon Steels" or "Steel - Carbon", which are synonym terms. Therefore, the SQL query generated by GPT-4o was not necessarily incorrect but incomplete. The prompt should include instructions regarding synonymous terms in the database, and the ground truth SQL query should be reformulated. The second wrong SQL query referred to counting the number of objects with fewer than 30 distinct properties. GPT-4o failed to return the total number of objects, but instead counted each object that had fewer than 30 distinct properties.

Llama 3.1 70B Instruct failed on seven SQL queries. The model used the wrong properties and values and selected two SQL templates incorrectly. It failed on the same issues as GPT-4o and generated only one invalid SQL query with a syntax error.

Llama 3.1 8B generated 45 incorrect SQL queries, 25 of which could not be executed in the database due to syntax errors. The LLM made several mistakes by translating property names from the database, which were in English, into Portuguese, disregarding the explicit instruction in the prompt. It also created names for properties that were not provided in the prompt, resulting in incorrect SQL queries. The examples added to the prompt helped the LLM craft SQL queries in the expected format, but the lack of attention to the use of properties affected the outcome.

Gemma 2 27B performed slightly better than Llama 3.1 8B. It generated 36 incorrect SQL queries, 19 of which could not be executed in the database due to syntax errors. It also made several mistakes by misusing property names, either translating or modifying them. It removed special characters that appeared in some property names, such as underscores and hash signs. Additionally, it incorrectly used the units of measurement provided in the prompt when attempting to extract numerical values from the *value* column, sometimes swapping millimeters with meters and using a Celsius degree symbol that is different from the one specified. Most of the time, Gemma was able to rely on the best SQL example given in the prompt to answer an NL question.

It is important to highlight that the accuracy of the models was much poorer before the removal of false negatives. GPT-4o achieved 62% while Llama 3.1 70B Instruct, Llama 3.1 8B, and Gemma 2 27B obtained 36%, 2% and 0%, respectively. Indeed, syntactically comparing the LLM-generated SQL query with the ground truth SQL query resulted in many false negatives – in several cases, the order of the filters in the WHERE clauses were different, as well as the aliases adopted in the JOIN clauses.

Finally, an important point to consider is that Llama 3.1 8B and Gemma 2 27B were downloaded in their Q4_0 quantization version to enable execution on a personal computer, which significantly impacts the quality of their results. Other quantization levels that reduce the loss of quality were tested, but they proved to be unfeasible due to the extreme slowness of model processing on the hardware used, as well as frequent CPU memory overflows, consuming 100% of the RAM.

Table 4: Results of experiments.

LLM Model	Duration[a]	Hits	Errors	Accuracy
Llama 3.1 8B	00:21:22	5	45	10%
Llama 3.1 Instruct 70B	-	43	7	86%
Gemma 2 27B	01:19:12	14	36	28%
GPT-4o	00:01:03	48	2	96%

[a] Ellapsed time to run all 50 questions in hours, minutes and seconds.

7 CONCLUSIONS

This paper addressed the text-to-SQL task for one very specific type of real-world relational database storing data extracted from engineering CAD files. The paper introduced a prompt strategy tuned to the text-to-SQL task over such databases, which was essential to achieve good performance. The paper concluded with a comparative analysis between different LLMs for executing the text-to-SQL task using real-world data from an oil and gas company.

The analysis of the experimental results showed that GPT-4o was the most accurate model, achieving 96% accuracy after removing false negatives, with only two errors in 50 NL questions processed. One of these errors revealed a flaw in the SQL ground truth itself.

On the other hand, among the open-source models, Llama 3.1 70B Instruct achieved the best accuracy with 86% showing a competitive result. The quantized versions of Gemma 2 27B and Llama 3.1 8B had very modest performances, although there was a significant improvement after removing false negatives. Gemma 2 27B's accuracy reached 28%, surpassing Llama 3.1 8B, which achieved only 10%.

The limitations of these quantized open-source models, especially regarding their ability to generate correct, syntax-error-free SQL queries, can be attributed in part to the quantization level required to run them on lower-capacity hardware. This suggests that the performance could be improved by using models with other quantization levels that reduce precision loss or even non-quantized models. Naturally, running open-source LLMs without quantization demands a better computational infrastructure, which leads to higher costs.

The main errors in the open-source models were due to incorrect property name translations, creation of nonexistent properties, and mishandling of special characters and units. Beyond prompt examples, success depends on clear question formulation, prompt adjustments, model size, configuration, and quantization level, especially for open-source models.

The prompt strategy proposed in this paper can be adapted to other domains and applications that require storing objects and their properties as key-value pairs. Future work might include conducting new experiments along many lines. First, new experiments with non-quantized open-source models should be conducted, in particular to test the limitations of the context window, albeit it was observed in the experiments that the average size of the prompts was just 8K and did not exceed the models' capacity. Second, an enhanced prompt should be tested to handle categorical attributes with synonymous values, possibly using a knowledge graph to guide the LLM in dealing with such situations. Finally, the experiments should be expanded to cover a combination of engineering data with other types of data.

ACKNOWLEDGEMENTS

This study was financed in part by CAPES – Finance Code 001, grants 88881.310592-2018/01, 88881.134081/2016-01, and 88882.164913/2010-01; by FAPERJ, grant E-26/204.322/2024; by CNPq, grant 305.587/2021-8; and by the Libra Consortium (Petrobras, Shell Brasil, Total Energies, CNOOC, CNPC, and PPSA), under an ANP research and development project.

REFERENCES

Avila, C. V. S., Vidal, V. M., Franco, W., and Casanova, M. A. (2024). Experiments with text-to-SPARQL based on ChatGPT. In *2024 IEEE 18th International Conference on Semantic Computing (ICSC)*, pages 277–284. https://doi.org/10.1109/ICSC59802.2024.00050.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. In *Proceedings of the 34th International Conference on Neural Informa-*

tion Processing Systems, NIPS '20, pages 1877–1901, Red Hook, NY, USA. Curran Associates Inc. https://dl.acm.org/doi/abs/10.5555/3495724.3495883.

Campos, J. G., De Almeida, V. P., De Armas, E. M., Da Silva, G. M. H., Corseuil, E. T., and Gonzalez, F. R. (2023). INSIDE: An Ontology-based Data Integration System Applied to the Oil and Gas Sector. In *Proceedings of the XIX Brazilian Symposium on Information Systems*, SBSI '23, pages 94–101, New York, NY, USA. Association for Computing Machinery. https://doi.org/10.1145/3592813.3592893.

Chang, S., Wang, J., Dong, M., Pan, L., Zhu, H., Li, A. H., Lan, W., Zhang, S., Jiang, J., Lilien, J., Ash, S., Wang, W. Y., Wang, Z., Castelli, V., Ng, P., and Xiang, B. (2023). Dr.Spider: A Diagnostic Evaluation Benchmark towards Text-to-SQL Robustness. *arXiv preprint*. https://doi.org/10.48550/arXiv.2301.08881.

Chiang, W.-L., Li, Z., Lin, Z., Sheng, Y., Wu, Z., Zhang, H., Zheng, L., Zhuang, S., Zhuang, Y., Gonzalez, J. E., Stoica, I., and Xing, E. P. (2023). Vicuna: An open-source chatbot impressing GPT-4 with 90%* ChatGPT quality. https://lmsys.org/blog/2023-03-30-vicuna/.

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. https://doi.org/10.1007/978-3-031-68309-1_8.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *arXiv preprint*. https://doi.org/10.48550/arXiv.1810.04805.

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*. https://doi.org/10.48550/arXiv.2308.15363.

Lei, F., Chen, J., Ye, Y., Cao, R., Shin, D., Su, H., Suo, Z., Gao, H., Hu, W., Yin, P., Zhong, V., Xiong, C., Sun, R., Liu, Q., Wang, S., and Yu, T. (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.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, pages 9459–9474, Red Hook, NY, USA. Curran Associates Inc. https://dl.acm.org/doi/abs/10.5555/3495724.3496517.

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. C., 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. https://dl.acm.org/doi/abs/10.5555/3666122.3667957.

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.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. In *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'14, pages 3104–3112, Cambridge, MA, USA. MIT Press. https://dl.acm.org/doi/10.5555/2969033.2969173.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E. H., Le, Q. V., and Zhou, D. (2024). Chain-of-thought prompting elicits reasoning in large language models. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, pages 24824–24837, Red Hook, NY, USA. Curran Associates Inc. https://dl.acm.org/doi/10.5555/3600270.3602070.

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. (2019). Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. *arXiv preprint*. https://doi.org/10.48550/arXiv.1809.08887.