Large Language Model for Querying Databases in Portuguese

Lourenço Figueiredo[1][0009−0009−6906−4495], Paulo Pinheiro[2,3][0000−0002−8912−2244], Luís Cavique[3][0000−0002−5590−1493], and Nuno Marques[14][0000−0002−3019−3304]

[1] Universidade Nova de Lisboa, FCT, Portugal
figueiredo.lourenco29@gmail.com
[2] CEDIS, Portugal
ppinheiro@cedis.pt
[3] Universidade Aberta and Lasige, FCUL, Portugal
luis.cavique@uab.pt
[4] NOVA LINCS, NOVA School of Science and Technology, Portugal
nmm@fct.unl.pt

Abstract. This study introduces a system that helps non-expert users find information easily without knowing database languages or asking technicians for help. A specific domain is explored, focusing on a subscription-based sports facility, which serves as an open-source version of a real case study. Utilizing the star schema, the available data in the database is structured to provide accessibility through Portuguese Natural Language queries. Using a Large Language Model (LLM), SQL queries are generated based on the question and the provided star schema. We created a dataset with 115 highly challenging questions drawn from real-world usage scenarios to validate the correctness of the system. Challenges found during testing, like attribute value interpretation, out-of-scope questions, and temporal interval adequacy issues, highlight the insufficiency of the star schema alone in providing the needed context for generating accurate SQL queries by the LLM. Addressing these challenges through enhanced contextual information shows significant improvement in query correctness, with validation results increasing from 57.76% to 88.79%. This study shows the potential and limitations of LLMs in generating SQL queries from Portuguese Natural Language queries.

Keywords: Natural Language Processing · Natural Language to SQL · Large Language Model · GPT-4 Turbo · Sports Facility Management · Databases.

1 Introduction

In our ever-evolving society, we have found a need to save and manage relevant data. In every domain, there is always some information we need to keep, either because we know it is relevant or because we predict it might be.

Historically, we have stored information physically. We would write in books and paper to prevent the loss of knowledge, though there would be some concerns. Information could very well be lost forever if its physical storage were to be ruined. Not only that but books or papers are hardly accessible to more than the one person who is reading them.

With the rise of computers and the digital world, it was found that information could very easily be saved digitally as data. Digital information can easily be copied and accessed by other readers. One file can be used by multiple users that do not even have to be in the physical location of its storage [5]. This, in turn, led us to store a great amount of information digitally.

Databases, which are organized collections of structured information [2], are created using programming languages with which we handle all of the data we want. We could now have access on demand to our information, but we are still bound by the programming language utilized in the database. This means anyone without a working knowledge of said programming language cannot view their desired information by themself, they require the help of a knowledgeable technician, whose time is better suited for other, more complex, challenges instead of being on call to translate information requests into database queries. The way to automate this process is through Natural Language Processing (NLP).

The motivation for this study is to innovate Sports Facilities Management by providing a simpler way of viewing their information through the use of an LLM. In Portugal, Sports Facilities currently deploy some Machine Learning algorithms to calculate and display relevant indicators but that is the full extent to which they dive in AI. In this paper, we aim to explore the performance of LLMs in facilitating user access to databases. A benchmark and several validations are proposed to check if a user that does not know SQL is capable of using the system to access information. Given we want to facilitate access to information, we are only exploring the SELECT command in this work.

In this section, the context and motivation for this work are introduced, as well as its objectives and case study. Following this, Section 2 delves into the disclosure of available tools and frameworks. Section 3 presents the proposed architecture of our system. Subsequently, Section 4 reports our findings, validates them, and discusses them. Finally, in Section 5, we report our conclusions.

2 Related Work

Natural Language Processing (NLP) enables machines to understand and generate human language, facilitating user interactions by automating tasks or providing assistance. A key application is translating natural language questions into SQL queries, known as NL2SQL. NL2SQL allows users to access database information without needing SQL expertise, removing a significant barrier to data retrieval. By using NLP techniques, we aim to make database querying more intuitive and accessible [3, 13, 14].

Recent advancements in NLP models have significantly improved NL2SQL systems, enhancing accuracy and reliability. These models understand and gen-

erate complex SQL queries from natural language input by tackling core challenges like intent understanding, schema mapping, and syntax generation [19]. Approaches such as sequence-to-sequence models, transformer-based architectures and attention mechanisms have been explored to boost NL2SQL performance[18]. Incorporating contextual information and domain-specific knowledge further refines accuracy, making these systems more robust and versatile. By addressing these challenges, NL2SQL systems empower non-expert users to interact with databases intuitively, improving accessibility and enabling users to harness the full potential of their data [15].

Large Language Models are deep learning language models pre-trained on an extraordinarily big corpus of data from books, chunks of the internet and any other source of information and their focus is to have the broadest set of knowledge possible, in many domains, as a human does. Generically, Language Models are computational models that utilize probability distributions over words in order to predict the next word in a sentence [6]. Over the years, initial Language Models have been growing in capability and size, with some of the latest and most popular Language Models now being referred to as Large Language Models(LLM) [11]. Language Models all currently base themselves on the Transformer architecture [17], an architecture proposed in 2017 that outperformed the former recurrent and convolutions architectures.

Benchmarks are datasets of input and output pairs that delineate the correct functioning of a language model. Like any other application or tool, language models also need a way to be evaluated and compared to each other. The way to do this is through benchmarks.

The Spider benchmark [20], for example, is a multi-domain dataset with 10181 questions on 200 different databases and is considered to be one of the best benchmarks in NL2SQL. However, the spider benchmark is available in English and has no context. The Spider benchmark allocates questions to a certain degree of difficulty, depending on what keywords and rules constitute their correct SQL query. Spider's Hardness Criteria for SQL queries defines 3 sets of clauses. The first include the SQL components `WHERE`, `GROUP BY`, `ORDER BY`, `LIMIT`, `JOIN`, `OR`, `LIKE` and `HAVING`. The second set includes the SQL components `EXCEPT`, `UNION`, `INTERSECT` and `NESTED QUERY`. The first two sets contain elements that might appear in an SQL query, with the second set also containing the possibility of there being a nested query. The third set contains conditions on if something happens more than once. These are the number of aggregating functions, the number of select columns, the number of where conditions, the number of group by clauses and the number of order by clauses.

The Spider Hardness Criteria for SQL queries three sets are then used to define four classification criteria for questions. `EASY` questions demand that there be **at most 1** keyword from the first set, **no** keywords from the second and **no** conditions satisfied in the third. `MEDIUM` questions demand that there either be **at most 1** keyword from the first set, **no** keywords from the second and **at most 2** conditions satisfied in the third **OR at most 2** keywords from the

first set, **no** keywords from the second and **at most 1** condition satisfied in the third. The criteria for `HARD` questions demand that there either be **at most 2** keywords from the first set, **no** keywords from the second and **at least 2** conditions satisfied in the third **OR exactly 3** keywords from the first set, **no** keywords from the second and **at most 2** conditions satisfied in the third **OR at most 1** keyword from the first set, **exactly 1** keyword from the second and **no** conditions satisfied in the third. Finally, the criteria for `VERY HARD` questions is every question whose SQL query is not captured in the other levels.

Challenges exist in any system in NL2SQL [15, 12]. These systems have to be aware of the ambiguity of Natural Language, how a word can have more than one meaning, how a sentence can have different interpretations and how the user may utilize one term. A simple Natural Language query could convert to a surprisingly more complex SQL query so the system should also be able to handle that [12].

GPT-3.5 and GPT-4 models are LLMs trained on vast internet data to predict the next token in a document, achieving human-like performance across various tasks, including code generation [7, 4, 16]. Other LLMs, like the openly available LLama3's [1], the Mistral models [10] and the Gemini models [8] are also quite capable but do not reach the heights of the GPT-4 models' performance and popularity. In our tests, we prompt the GPT-4 model with Natural Language queries to translate them into SQL queries for database access. However, the accuracy of the system depends on the contextual information provided in the prompt. To improve accuracy, complex table and attribute names are simplified for better model comprehension. API calls to a specific GPT-4 Turbo model allow for conditioning its behavior and improving determinism in replies[5]. This approach improves the consistency and the reproducibility of the results in our experiments.

3 Proposed Model

Our system is a Natural Language Interface for a Database and its architecture can be seen in Figure 1. It consists of a Large Language Model, a pre-defined context to fine-tune it and a Query Execution Component. The system interacts with a Database and an Interface, which handles all of the interactions with the User.

A User interacts with the system by providing, in the Interface, a Natural Language Query to which the User wants to know the information to. The question should be clear and concise, in order not to confuse the model, as mentioned in the previous section.

The pre-defined context, represented in Figure 2, is a string of text containing all the information we need to supply to the model along with the question in the prompt. This string contains the star schema of the slice of the database we're

[5] https://platform.openai.com/docs/api-reference/chat/create

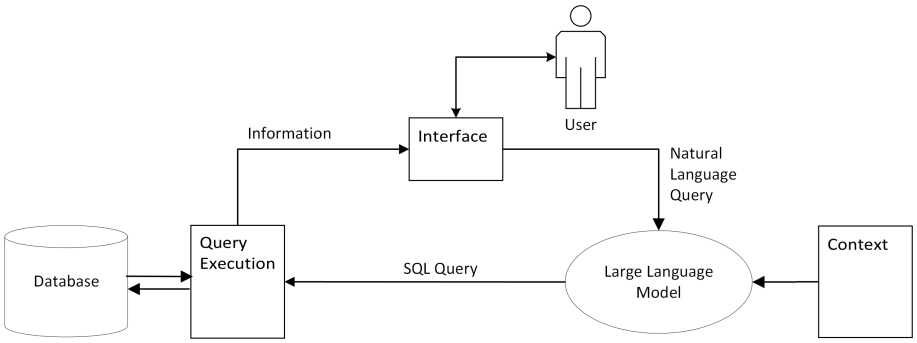

Fig. 1. Our proposed generalized system architecture (please check text for specific details).

utilizing from our case study and phrases that better contextualize everything that's accessible in this slice.

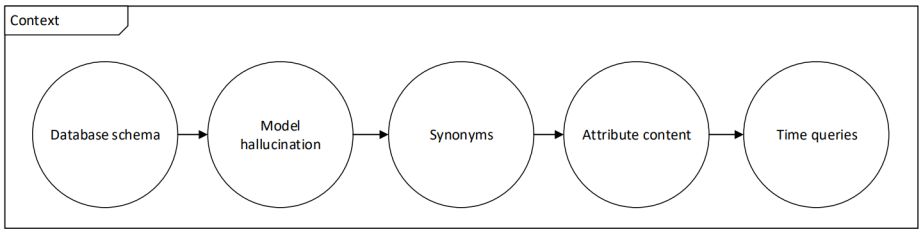

Fig. 2. Context representation. arrows indicate the order they were inserted in, not correlation.

The LLM used in our system, GPT-4 Turbo (`gpt-4-0125-preview`), was selected for its renowned state-of-the-art performance and user-friendliness. The LLM takes both the Natural Language query and the Context and generates a Transact SQL query, as that is the SQL language used in our case study.

The database used in the system is provided by the case study and it is proprietary. Any and all use of that database's attributes is masked with fake attribute names. The Query Execution component runs the obtained SQL query on the database and sends the retrieved information to an external Interface, which then presents the data to the User.

The SQL query obtained from the model should be reliable and the expected response to a certain question. By this, we mean that we want to achieve determinism in our responses, or at least minimize the inherent randomness in working with Large Language Models. To do this, we can utilize the model's

seed and temperature parameters[6], which we previously mentioned. By setting a seed, we are instructing the model to, theoretically, pick the same sequence of tokens if confronted with the same prompt. This is not foolproof, as OpenAI relates, the seed parameter does not force determinism, only brings the model closer to it. By setting the sampling temperature to 0, the model will also lean more into deterministic and focused outputs. It is also not foolproof but we will settle for as close to deterministic outputs as possible.

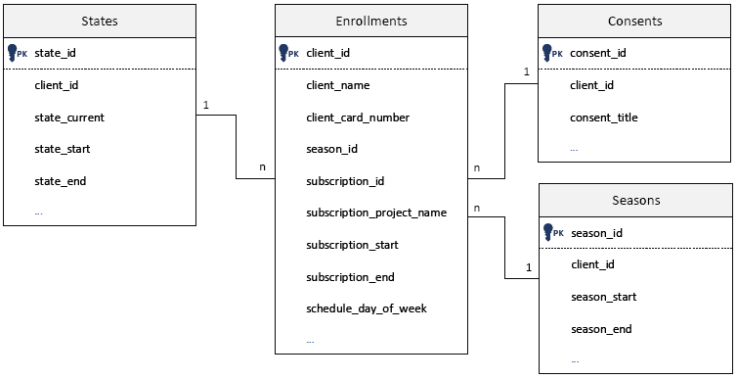

Fig. 3. A simple and attribute-hiding representation of the database slice's star schema.

We are using a slice of a database from our case study. The data is organized in a star schema with a view acting as our table of facts and three other views being our dimension tables, as seen in Figure 3. The enrollments view contains information on clients, subscriptions, projects, schedules and more. The status view adds information about clients' status, whether they are active users, dropout users and more. This view connects to the star view through the `client_id` attribute. The consents view adds information about what the clients have consented to. This view connects to the star view through the `client_id` attribute as well. The seasons view adds information about custom intervals of time. This view connects to the star view through the `season_id` attribute. The full range of attributes is not shown in this paper to safeguard the case study's privacy and system safety. The attributes were given more suggestive names than their original names to improve readability and model comprehension.

4 Results and Validation

We developed a system capable of generating SQL queries from the Portuguese Natural Language in Portuguese Databases. To properly analyze the system, we ran 2 experiments.

[6] https://platform.openai.com/docs/api-reference/chat/create

4.1 Experiment 1: Synonyms

The first experiment involved observing how the GPT-4 Turbo model understands concepts represented in our database. This revolved around asking the model what it understands by the name of an attribute in our current context and seeing whether it has the same definition for the word as we do.

We wanted to determine if the model was capable of correctly determining which words are synonyms to the names of certain attributes. For example, the star schema has attributes with data on clients, but questions involving individuals, users or partners should also refer to the same attributes. In Portuguese, `usuário`, `utente` and `utilizador` all translate to `user` and they all should refer to the `client` attributes. Should the model not provide us with a convincing definition, then we'd need to specify additional information about them and declare that certain words are, in fact, synonyms.

In another example, by asking the model what it understands by the term `localidade` (`locality-place`), we get a definition that aligns with what we want the model to know. We repeated this for other attributes whose names could have ambiguous meanings. From definitions that did not align with what we wanted, we learned which attributes needed additional information to provide their meaning. We then compared the definitions to their synonyms' to see if the model gives them the same meaning, considering them synonyms. To conclude the experiment, we asked the model questions using synonyms instead of attribute names to see if the model would use the correct attributes. The results are as shown in Table 1. In the first row, we measure the ratio of success to which the model gave a good definition for every attempted word whose definition was asked of the model. In the second row, we measure the ratio of success to which the model recognized two words as synonyms in the domain by asking questions with synonyms of the attribute names and checking if the model chose the right attributes in the SQL query. In the third row, we repeated the measurements in the second row but with improved context detailing the synonymy between the known words in the case study. It is worth noting the model did not fail when explicitly told this information. This evaluation was performed manually by the first author and reviewed by the other three.

Table 1. The rate of acceptance for each word's meaning and synonyms.

Category	Correctness
Word Meaning	0.6429
Word Synonym	0.7500
Word Synonym with Context	1.0000

To avoid any confusion, the meaning of a word being the same as another should make them synonyms but we are reporting on what we found using this model. The model, without additional information, would give somewhat differing answers on the definitions of those words. However, the model was capable of

using the right attributes when asked to generate SQL queries from Portuguese Natural Language questions more times than what the results on Word Meaning showed, as seen in Table 1. We then added to the context information about the meaning of certain words and what their synonyms are.

4.2 Experiment 2: Questions

For our project, we created our own benchmark (`ESPORTNL2SQL`) to generate more domain-specific questions, as the Spider benchmark is too broad and not tailored to our case study's database. `ESPORTNL2SQL`was created from 115 real-world questions used regularly in the case study. Our benchmark presents biased results given the niche of the domain we worked in, so we decided to use the Spider's Hardness Criteria for SQL queries. We added a new difficulty level called `Wrong NL` to represent cases where the model should not generate an SQL query when the question is not allowed or out of scope. We also exchanged `LIMIT` with `TOP`.

As seen in Table 2, the GPT-4 Turbo model is capable of generating correct SQL queries given their corresponding Natural Language question. The first row measures the rate of correctness of the model on our benchmark when only given the star schema of the database. The second row measures the rate of correctness of the model on our benchmark when given the entire context that further explains the star schema. By giving the model this context, the model becomes more reliable in generating correct SQL queries. The validation was done in a spreadsheet that is represented by a small extract in Table 3. This validation was performed by the first author, with the remaining three having reviewed it. Some information is hidden to preserve its sensitivity[7]

Table 2. Validation comparison between contexts

Model	Easy	Medium	Hard	Extra Hard	Wrong NL	Total
GPT-4 Turbo (only star scheme)	0.7826	0.5882	0.7000	0.5500	0.0833	0.5776
GPT-4 Turbo (context beyond star scheme)	0.9565	0.9412	0.8000	0.6500	1.0000	0.8879

For a more qualitative description, with the context of the database given to the model being the star schema and the additional information on what some attributes mean, the model was asked several questions to discover what other problems there were.

Several authors report model hallucination problems [9]. This problem was observed in our queries with the model's insistence on choosing attributes from the view but trying to get them from fictitious tables, as seen in Figure 4(A). The

[7] Additional commented illustrative examples and the full set of questions are available at `https://bit.ly/4bJ3cbs`

Table 3. Extract from Validation Spreadsheet

Question	SQL Hardness	SQL Correctness
Was João enrolled in swimming in January?	Medium	correct
How many clients enrolled in fitness for the first time in January?	Very Hard	correct

star schema provided in the context does not mention a table `schedule_details`, but as several attributes' names start with that name, the model assumes that table exists and can be accessed, which also creates a security problem since the table could exist and the database structure would be leaked. This delusion is not specific to this imaginary table, it could also be found in questions about clients or subscriptions. There were instances where the model would generate the correct query but not correctly select the name of the view. The view is called `enrollments`, yet the model would try to select from `view_enrollments`, as seen in Figure 4(B,D). The problem is solved by adding to the context that the model should only utilize the described views and nothing more.

The Attribute Content problem arises when a model cannot understand the meaning of certain attribute values. While the model can identify data types and often deduce likely values, specific attributes like gender or marital status are not automatically inferred. For instance, a `VARCHAR(1)` attribute for gender may allow an LLM to predict characters for male and female, but not for other gender representations. With status attributes, the model relies on training data to infer meanings. For `status_current` in Portuguese, the model correctly assumes that active users are represented by a value of 1 and dropout users by 3. However, it fails to infer other states and sometimes incorrectly compares `VARCHAR` attributes with state names. To resolve this, each possible attribute value and its context-specific meaning must be enumerated. This eliminates blind guessing and enables the model to generate accurate queries, as illustrated in Figure 4(C).

The Time Queries Problem is related to time in more than one way. To start, if the model was asked to provide an SQL query about something occurring until today or between two months, it sometimes would use a specific date instead of using a function called `GETDATE()` to get that day's date and others to get the months in the same year as the date, as seen in Figure 4(D). This would not be a problem if the model did not assume we are currently in the year 2023. This happens due to the model's training data being only up to December 2023 [8]. From testing, it seemed like the model would use the `GETDATE()` function only sometimes when it needed to know the exact date or when it needed to know the year or month we are currently in. The first attempt at fixing this problem went through telling the model explicitly to use the `GETDATE()` function to get today's date whenever needed. This did not work as the model kept trying to use a literal wrong date as today's date. The easier fix was to then just give the current year in the context. By telling the model we are currently in the year 2024, it no

[8] https://platform.openai.com/docs/models/continuous-model-upgrades

longer tries to use 2023 for dates in this year. Moreover, another problem has to do with time intervals. If the system user wants to know about clients that had a subscription in a certain class between any two dates, then it should look for subscriptions that started before the end of the class and ended after the beginning, as seen in Figure 4(E). As long as a class does not start after it ends, this is the easiest condition to check if an interval of time intersects with another. The model, however, kept trying to use the starting time of the interval before the beginning of the class and the end of the interval after the end of the class or vice-versa. In Figure 4(D), the model correctly selects subscriptions ending after the interval's beginning, but incorrectly only selecting subscriptions that start in the interval, effectively removing any entries that start before it even though it is a subscription to a class that intersected that time period. There was, in fact, a client subscribed to a class at some point during the considered time interval. The only way to fix this problem was to tell the model exactly how to proceed in these situations, but it sometimes still fails depending on the wording of the question, which is fine, as this is a specific fix for our case study.

5 Conclusions and Future Work

Our study introduces a system for generating SQL queries from the Portuguese Natural Language within the framework of a company's production sports database, validated through real-world client inquiries. We evaluated its performance and identified challenges in an enterprise setting. Experiment 1 examined the LLM's synonym comprehension, revealing the necessity of contextual cues for refining query accuracy. Experiment 2 highlighted challenges such as model hallucination and time interval handling, emphasizing the importance and relevance of precise and model-tuned context. By utilizing Spider SQL-benchmark Hardness Criteria, we assessed the system's efficacy against intricate queries evaluating system proficiency before and after additional contextual input is provided.

This work highlights the potential of LLM-based systems for effective database querying in real-world business settings. The results give evidence of the critical role of context in enhancing query accuracy, especially in handling text ambiguities inherent in natural language input. By improving the context given to the LLM, the performance improves, as we saw in this domain. NL2SQL systems must adeptly navigate these ambiguities to generate correct SQL queries, ensuring user satisfaction and system effectiveness. Despite the complexity that may arise from converting simple natural language queries into more intricate SQL queries, NL2SQL systems maintain their user-friendly nature, aligning with the universal familiarity with natural language communication. Moreover, their simplicity enhances information accessibility, a key advantage emphasized in this study. Additionally, these systems exhibit a degree of tolerance for grammatical errors, as they can decipher the intended meaning of prompts even with minor errors that do not alter the overall context. The cost of queries with the added context was usually double the cost of queries without additional context. The developed system has several limitations. Although the benchmark was created

```sql
-- A) How many users are registered on Mondays?
SELECT COUNT(DISTINCT subscription_id) AS users_registered_monday
FROM shedule_details
WHERE shedule_details_day_of_week = 1 AND shedule_details_status = 1;

-- B) What are the clients with active status?
SELECT c.client_name FROM view_enrollments AS c
     JOIN states AS e ON c.client_id = e.client_id
WHERE e.current_state = 1; -- Assuming 1 represents an active state

-- C) What are the clients in pre-initiation state?
SELECT DISTINCT client_name, client_card_number FROM states
    JOIN enrollments ON states.client_id = enrollments.client_id
WHERE state_current = 0;

-- D) Who is enrolled in swimming classes between January and February?
SELECT DISTINCT client_name FROM view_enrollments
WHERE subscription_project_name LIKE '%swimming%'
      AND subscription_start BETWEEN '2023-01-01' AND '2023-02-28'
      AND subscription_end >= '2023-01-01';

-- E) Was João enrolled in swimming in January?
SELECT DISTINCT client_name, client_card number FROM enrollments
WHERE client_name LIKE '%João%'
      AND subscription_project_name LIKE '%swimming%'
      AND subscription_start < '2024-02-01'
      AND (subscription_end > '2024-01-01' AND
        subscription_cancel_in > '2024-01-01');
```

Fig. 4. Examples of the model: A) trying to select data from non-existent or non-accessible tables. B) assuming what the value of an attribute means. C) generating the right query because it was given additional information D) choosing the wrong year and time interval. E) correctly choosing the right time interval.

using real questions from actual clients, it does not fully encompass the entire domain of Sports Facility Management. For instance, areas such as Billing and Access Control are not represented in our star schema. Furthermore, the benchmark and its results cannot be fully disclosed due to the inclusion of privileged information. Moving forward, future research will further explore and address these challenges to continually refine NL2SQL system performance and usability in real-world scenarios and further experiments will measure and try to improve the cost of LLM usage in this scenario. The next step is using newer, improved LLMs, a more comprehensive star schema of a database and additional SQL retrieval commands.

References

1. Llama3 blog, https://ai.meta.com/blog/meta-llama-3/
2. What is a database?, https://www.oracle.com/database/what-is-database/

3. What is natural language processing (nlp)?, https://www.ibm.com/topics/natural-language-processing
4. Brown, T.e.a.: Language models are few-shot learners. In: Larochelle, H., Ranzato, M., Hadsell, R., Balcan, M., Lin, H. (eds.) Advances in Neural Information Processing Systems. vol. 33, pp. 1877–1901. Curran Associates, Inc. (2020)
5. Butler, M.A.: Issues and challenges of archiving and storing digital information: Preserving the past for future scholars. Journal of library administration **24**(4), 61–79 (1997)
6. Chang, Y., et al.: A survey on evaluation of large language models. ACM Trans. Intell. Syst. Technol. (jan 2024), https://doi.org/10.1145/3641289, just Accepted
7. Deng, J., Lin, Y.: The benefits and challenges of chatgpt: An overview. Frontiers in Computing and Intelligent Systems **2**(2), 81–83 (2022)
8. Gemini Team, et al.: Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context (2024), https://arxiv.org/abs/2403.05530
9. Huang, L., et al.: A survey on hallucination in large language models: Principles, taxonomy, challenges, and open questions (2023)
10. Jiang, A.Q., et al.: Mistral 7b (2023), https://arxiv.org/abs/2310.06825
11. Kaplan, J., et al.: Scaling laws for neural language models (2020)
12. Katsogiannis-Meimarakis, G., Xydas, M., Koutrika, G.: Natural language interfaces for databases with deep learning. Proc. VLDB Endow. **16**(12), 3878–3881 (aug 2023), https://doi.org/10.14778/3611540.3611575
13. Khurana, D., Koli, A., Khatter, K., Singh, S.: Natural language processing: State of the art, current trends and challenges. Multimedia tools and applications **82**(3), 3713–3744 (2023)
14. Liddy, E.D.: Natural language processing (2001)
15. Majhadi, Khadija, Machkour, Mustapha: The history and recent advances of natural language interfaces for databases querying. E3S Web Conf. **229**, 01039 (2021), https://doi.org/10.1051/e3sconf/202122901039
16. OpenAI, J.A.e.a.: Gpt-4 technical report (2023)
17. Vaswani, A., et al.: Attention is all you need. In: Guyon, I., Luxburg, U.V., Bengio, S., Wallach, H., Fergus, R., Vishwanathan, S., Garnett, R. (eds.) Advances in Neural Information Processing Systems. vol. 30. Curran Associates, Inc. (2017)
18. Xu, K., et al.: Graph2seq: Graph to sequence learning with attention-based neural networks (2018)
19. Y., S.L., et al.: Natural language to sql: Automated query formation using nlp techniques. E3S Web Conf. **391**, 01115 (2023), https://doi.org/10.1051/e3sconf/202339101115
20. Yu, T., et al.: Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task (2019)