Available online at www.ijournalse.org

Emerging Science Journal

(ISSN: 2610-9182)

Vol. 8, No. 5, October, 2024

Zero-Shot Prompting Strategies for Table Question Answering with a Low-Resource Language

Marcelo Jannuzzi [1], Yuriy Perezhohin [1], Fernando Peres [1], Mauro Castelli [1*], Aleš Popovič [2]

[1] *NOVA Information Management School (NOVA IMS), Universidade NOVA de Lisboa, Campus de Campolide, 1070-312, Lisboa, Portugal.*

[2] *Faculty of Computer and Information Science, University of Ljubljana, Ljubljana, Slovenia.*

Abstract

This work explores the application of zero-shot prompting strategies for table question answering (TQA) in Portuguese, focusing specifically on the Text2SQL task. This task involves translating questions posed in natural language into Structured Query Language (SQL) queries, which can be executed against a database to answer the original question. Given the popularity of relational databases across various domains, advancements in this field can substantially impact the accessibility and democratization of data as simpler and more intuitive interfaces for database interaction are developed. Despite this significant potential, progress in developing Portuguese TQA solutions remains limited. The proposed approach leverages Large Language Models (LLMs)—specifically the GPT-3.5 and GPT-4 models—through zero-shot prompting. The primary objectives are to assess the effectiveness of such LLMs in this task and to identify the most suitable prompt styles. These are evaluated using a Portuguese translation of the popular Spider Text2SQL benchmark. Results reveal that the proposed approach can generate adequate SQL queries to answer Portuguese language questions about various databases, mainly when using GPT-4. The findings suggest that including schema information and database content in the prompts is critical for satisfactory outcomes.

Keywords:

Text to SQL;

Natural Language Processing;

Computational Linguistics;

Zero-shot Prompting.

Article History:

Received:	20	May	2024
Revised:	14	September	2024
Accepted:	19	September	2024
Published:	01	October	2024

1- Introduction

A vast amount of the world's information resides within databases. These repositories store and retrieve both structured and unstructured data, serving a wide range of applications. However, accessing this information often presents a challenge for non-technical users. Interacting with databases frequently requires knowledge of query languages like SQL, along with an understanding of the data's structure and organization. This reliance on technical expertise or pre-built interfaces can be a significant barrier. Even for technical users, formulating questions to extract specific information can be a time-consuming and potentially error-prone process. Thus, there is a need for tools that facilitate simpler and more natural interactions with database information. For relational databases specifically, a crucial step lies in translating natural language questions into executable SQL queries. This task, known as Text2SQL, allows users to retrieve the desired information directly from the database.

The Text2SQL task has been an area of active research for several years [1–4], accompanied by the significant evolution in the field of Natural Language Processing (NLP) tasks. With the rise of deep learning and the development of the Transformer architecture [5], coupled with the growing popularity of Large Language Models (LLM), the Text2SQL task has seen renewed interest [6–8]. The development of LLMs, particularly models like GPT [9–11], T5

* **CONTACT**: mcastelli@novaims.unl.pt

DOI: http://dx.doi.org/10.28991/ESJ-2024-08-05-020

[12], and BLOOM [13], is especially interesting. These models have demonstrated the ability to solve various tasks without extensive fine-tuning [9, 14]. Moreover, they have also introduced a novel interaction method called prompting, where the task to be performed and any relevant context are described in natural language.

However, similar to other NLP tasks, the majority of research in Text2SQL has focused on English, with limited work dedicated to low-resource languages like Portuguese [15, 16]. Notably, the application of prompting techniques enabled by LLMs remains unexplored for Portuguese. This is the gap that this work aims to address. This work proposes a novel approach to the Text2SQL task for Portuguese using LLMs. We use a Portuguese translation [16] of the Spider Text2SQL benchmark [17] to evaluate the performance of GPT-3.5 [10] and GPT-4 [11] models on this task. The study considers the effectiveness of various zero-shot prompting styles (where the model receives instructions describing the task without any guiding examples) and investigates the impact of including information like database schemas and table content within the prompts.

This research contributes significantly to the advancement of knowledge in two crucial areas: NLP for low-resource languages and the application of LLMs to the Text2SQL task. In particular, by exploring zero-shot prompting for Portuguese Text2SQL, the study addresses a critical gap in current research, where the focus has primarily been on English. This holds significant implications for increasing accessibility and information retrieval for users who may not have fluency in English or lack technical expertise in query languages.

The remaining part of the paper is organized as follows: Section 2 provides the theoretical background and relevant related work on NLP, LLMs, and the Text2SQL task. Section 3 describes the methodology, including the data used, the experimental pipeline, and the evaluation metrics. Section 4 presents the main findings of the experiments, as well as a comparison to results available in the literature and an error analysis. Finally, Section 5 summarizes the main contribution of the paper and discusses future research avenues.

2- Theoretical Background & Related Work

2-1- Table Question Answering

Table Question Answering (TQA) is the task of extracting information from tables to answer questions posed in natural language. Given one or more tables and a question, the goal is to find the relevant information within the tables (if it exists) to provide a correct answer. The type of answer determines the specific type of TQA task. Free-form TQA involves answering questions in a conversational manner, where the answer may be a sentence or even a paragraph. In contrast, non-free-form TQA provides more direct answers, usually consisting of a single value or a few words [18].

Two main approaches exist for solving the TQA task: semantic-parsing-based methods and non-semantic-parsing-based methods [18]. Semantic-parsing approaches first translate the natural language question into a machine-readable format, such as a SQL query. This query can then be executed against the table to retrieve the answer. Methods within this approach range from rules-based solutions using hand-crafted features [2, 19] to reinforcement learning [20] and sequence-to-sequence models [21, 22]. In contrast, non-semantic-parsing methods learn to directly map the question to the answer without first translating it into a logical form. Examples of this approach include generative [23, 24], extractive [25, 26], matching-based [27, 28], and retriever-reader-based methods [29].

2-2- Text2SQL

Text2SQL is a specific type of TQA that employs semantic parsing. In this approach, a natural language question is transformed into a SQL query. This SQL query can then be executed against the target tables to retrieve the answer. Given the prevalence of relational databases in businesses, Text2SQL holds significant importance. It allows for the development of natural language interfaces for these systems. These interfaces allow non-technical users, unfamiliar with SQL, to interact with databases more intuitively and conversationally. Even for users proficient in SQL, the ability to query a database in natural language can be beneficial. It reduces the need to memorize complex database schemas, field names, and relationships between tables [3].

Given the relevance of the task, Text2SQL has been the subject of several studies that have proposed different approaches to the problem. One such approach is the rules-based one, which involves mapping words in the question to specific query components such as SQL clauses, table relations, and values [3, 4, 30]. Reinforcement learning-based approaches have also been proposed [20], where an agent acts sequentially to maximize a reward function based on the candidate answer, the question posed, and the associated table.

However, this area of research has seen significant activity with the advent of deep learning and the increased interest it has brought to the field of Natural Language Processing (NLP). Xu et al. [31] approached the problem as a slot-filling task where, given a question and a template SQL query, the objective is to fill each slot with the appropriate value. Yu et al. [32] built upon this by introducing a type-aware slot-filling approach, where the type of the question entities (derived from an external knowledge base) is used to guide the slot-filling process. Iyer et al. [33] and Gur et al. [34] proposed human-in-the-loop sequence-to-sequence models to improve the quality of the generated SQL queries from user interaction. Sun et al. [22] took advantage of the structure of tables and the syntax of SQL to generate more accurate and valid queries, and Bogin et al. [35] also leveraged the structure of the database schema, encoding it with a graph neural network.

The transformer architecture [5] and large pre-trained language models have also made significant contributions to Text2SQL. Initial solutions involved fine-tuning general-purpose language models like BERT [36] or T5 [12] on datasets containing questions, SQL queries, database schemas, and possibly database content. However, a drawback of these initial approaches with pre-trained models was their tendency to output invalid SQL code. To adhere to the rigid syntax of SQL, various methods to constrain the outputs of such models have been proposed [7, 37–39]. In particular, the RAT-SQL system [6] has attracted a lot of research and industry interest, achieving state-of-the-art performance on the Spider benchmark [17] at the time of its release. This system utilizes a relation-aware transformer to represent the database schema and the natural language question.

Furthermore, results from the broader field of general-purpose NLP suggest that simply increasing model size and training data may lead to advanced capabilities. This includes the ability to perform tasks with little or no additional training (few-shot and zero-shot learning, respectively) [9, 40]. These models can even exhibit novel, emergent abilities not present in smaller models [41], such as arithmetic, word unscrambling, and figure-of-speech detection. Within the Text2SQL domain, Rajkumar et al. [8] demonstrated that the Codex model [40], without any fine-tuning, achieves competitive performance on the Spider benchmark's [17] development set. Furthermore, it surpasses state-of-the-art fine-tuned models on benchmarks like GeoQuery [2, 42] and Scholar [33].

While research on Text2SQL for non-English languages is limited, there have been some recent advancements. Min et al. [43] adopted a neural sequence-to-tree method for Chinese, using an LSTM sequence encoder to encode the input question, outputting a SQL query in its syntactic tree form. For Vietnamese, Tuan Nguyen et al. [44] fine-tuned the pre-trained model PhoBERT [45]. In Portuguese, da Silva and Jindal [15] translated user questions from Portuguese to English before applying existing Text2SQL methods to generate queries. More recently, José & Cozman [16] adapted the RAT-SQL+GAP system [46] to a multilingual context. They accomplished this by producing a Portuguese-translated version of the Spider dataset [17] and using RAT-SQL+GAP to train different fine-tuned BERT [47] and BART [48] models. Interestingly, their findings show that fine-tuning a multilingual BART model on a combined dataset of Portuguese and English questions yielded better results than using only Portuguese questions.

2-3- The Evolution of NLP

The field of NLP has evolved significantly over the past decades. Five distinct paradigms have emerged, each building upon the previous: Heuristics-based, non-neural network fully supervised learning, neural network fully supervised learning, pre-train, fine-tune, and pre-train, prompt, predict [49]. Relied on manually crafted rules and domain-specific heuristics. They leveraged resources like dictionaries and thesauruses to understand word meanings, synonyms, and antonyms [50]. Regular expressions (and more generally, context-free grammars) have been widely used in building rules-based NLP systems and are still used today in closed domains where accuracy and completeness of coverage are particularly important [51]. With the popularization of machine learning, different classification and regression techniques were applied to text data just as they were applied to tabular, image, or audio data. However, due to both the nature of the algorithms available and the relatively small size of the existing datasets (which were often manually labeled), this paradigm relied heavily on hand-crafted features extracted from the raw text, which was necessary to guide such models in learning from the limited data [52–55].

As datasets increased in size and access to computing power became cheaper and more widespread, neural networks began gaining popularity in various machine learning tasks [56], particularly in image recognition, leading to what is now known as the "ImageNet moment" [57] in 2012. Applications in NLP also followed, particularly with the use of Convolutional Neural Networks (CNNs) [58, 59] and Recurrent Neural Networks (RNNs). With the advent of neural networks, the need for hand-crafted features was reduced as the models could learn from the raw text data itself, and the research focus shifted to designing architectures better suited to this. However, the reliance on labeled datasets to train these models remained.

Starting with the introduction of the Transformer architecture [5], the fully supervised paradigm began playing a decreasing role in NLP. Instead, researchers began pre-training their models on large unlabeled datasets of raw text and then fine-tuning them on smaller labeled datasets for specific tasks [60–62]. As researchers began standardizing on a few key model architectures, the focus in this paradigm shifted to objective engineering: designing the pre-training and fine-tuning training objectives [49]. These fixed architectures arose from the unsupervised pre-training phase allowing the models to learn general-purpose representations of language, and also due to the high costs required in pre-training.

As for the pre-training objectives, the most popular ones are masked language modeling and next token prediction (also referred to as autoregressive). In masked language modeling, popularized by the BERT language model [36], a fraction of the input tokens are masked, and the model must predict the original value of the masked tokens (BERT also makes use of a second pre-training objective, next sentence prediction, which involves predicting whether two sentences are consecutive in the original text). Next token prediction, popularized by the GPT family of models [60], involves predicting the next token in a sequence given the previous ones. BERT is a notable example of this paradigm, having played a key role in its popularization. Its base architecture has been fine-tuned on a variety of languages and tasks.

As language models became ever larger and trained on increasing amounts of raw data, it was found that they were able to solve a variety of tasks without any fine-tuning [9, 14], reducing, if not eliminating, the need for large labeled datasets. Rather than adapting pre-trained models to a task through objective engineering, tasks are modified to resemble that of the training objective through what is known as a prompt.

By crafting appropriate prompts, different behaviors can be elicited from the same model, which can then be used to solve a wide range of tasks, often without any additional task-specific data (in what is known as zero-shot learning). It is only needed to find the most appropriate prompt, which has introduced the need for prompt engineering. Models arising from this paradigm are often referred to as Large Language Models (LLMs), and important examples include BART [63], GPT-3 [9] (perhaps the most influential model of this paradigm), T5 [12], LaMDA [64], OPT [65], PaLM [66], and BLOOM [13].

2-4- Prompting

The latest development in NLP has introduced a novel way to interact with language models: *prompting*. A *prompt* is essentially a textual instruction that tells the model what task to perform and provides any relevant background information. This instruction takes the form of a template with two key elements:

1. ***Input slot ([X]):*** where the input sequence is inserted.
2. ***Answer slot ([Z]):*** this slot serves as a starting point for the model to generate an answer. This answer can be further refined through post-processing steps before becoming the final output.

Note that the answer slot may be either in the middle of the prompt or at the end. As shown in Table 1, the former is known as a *cloze prompt* and the latter as a *prefix prompt*. Also, the number of input and answer slots is not limited to a single one and may be changed to suit the task.

Table 1. Examples of cloze and prefix prompts

Type of prompt	Prompt template	Filled Template
Cloze prompt	The capital of [Z] is [X]	The capital of Portugal is Lisbon
Prefix prompt	The capital of [X] is [Z]	The capital of Portugal is Lisbon

It is important to note that prompts are not limited to natural language. They can be composed of programming language tokens, and some methods even construct prompts directly in the embedding space of the model. These are known as *soft* or *continuous prompts*. Continuous prompts are interesting because they remove the constraint that the prompt embeddings be that of natural language tokens, and, being in the embedding space, they can be more easily tuned to the task at hand.

Prompts constructed from phrases of discrete language tokens (natural or otherwise) are known as *hard* or *discrete prompts.*

2.4.1. Factors Affecting Prompting Approaches

There are three main factors to consider when using a prompting technique: the choice of language model, the design of the prompt itself, and how the final answer is processed (post-processing).

The choice of language model is critical to the success of a prompting approach, and there are many options from which to choose. Considerations when selecting an appropriate model for a task include:

1. ***Size***: Larger models with more parameters generally outperform smaller ones (although there are limitations) [8, 9, 40]. They can even exhibit entirely new capabilities not seen in smaller models [41]. However, larger models require more computational resources, making them more expensive to run.
2. ***Training Data***: The amount and type of data used to train the model are crucial. Models trained on more data typically perform better than those trained on less [67]. However, the type of data present in the model's training corpus is also important, as models trained predominantly on text from a specific domain (e.g., Wikipedia or social media posts) may not perform well on tasks in a different domain (e.g., medicine or code completion).
3. ***Training Objective***: Different objectives have been used to pre-train language models, and there is evidence that different objectives may be more suitable for specific tasks [68]. The autoregressive objective performs particularly

well on language generation tasks and is especially suited for interaction via prompting. However, its unidirectional nature limits it to generating text from left to right and hinders incorporating context from the right side of the input. Masked language modeling, on the other hand, can capture contextual information from both sides of the input sequence, usually achieving better performance when fine-tuned on downstream NLU tasks, such as classification [49, 69].

As previously mentioned, a prompt consists of a template string with empty slots to be filled, thus specifying the task to be performed. As such, the choice of template not only has a significant effect on model performance but also on what task the model is even *capable* of performing [70, 71]. The process of designing a high-performing prompt for a specific task is known as *prompt engineering*.

The same model can produce different results when prompted in different ways (sometimes even trivially different). Because of this, prompting has been described as a *brittle process* [72]. A simple change like capitalization or the order of information in the prompt can make the difference between random guessing and near state-of-the-art performance [70, 73].

The critical impact of prompt design on model performance is evident in the Text2SQL task [8]. When prompted with only the question (for which a SQL query was needed), the model achieved an 8.3% execution accuracy. Adding schema information increased this to 56.8%, and including a few database content rows further improved it to 67.0%. This highlights the importance of prompt engineering. Liu et al. [49] categorized various forms of prompt engineering explored in the literature to date:

1. ***Manual template engineering***: this is the most common and straightforward approach, involving manually designing prompt templates based on human intuition and understanding. It is often the first step when developing a prompt-based solution for a task. However, it can be time-consuming and lack a guarantee of finding the optimal prompt. Here are some notable strategies within this category:

(a) *Zero-shot prompting*: The model is simply given a natural language instruction describing the task, without guiding examples [9].

(b) *Few-shot prompting*: A few examples of the desired input-output pairs are provided, and the model is then prompted to generate the remaining example [9].

(c) *Colon prompting*: Usually used with few-shot prompting, the colon prompt format involves simply adding a colon to the end of the input sequence as a signal to the model that the next token should be generated [74].

(d) *Masterful-type prompting*: The prompt includes the context that a "master" on the subject will perform the task [74], such as a "masterful translator" for translation tasks, or a "masterful programmer" for code completion tasks.

(e) *Chain of thought prompting*: Usually used for question-answering tasks in a few-shot setting, the model is prompted to generate a sequence of intermediate reasoning steps before arriving at the final answer [75]. Forcing the model to decompose multi-step problems into intermediate steps can help it arrive at the correct answer.

(f) *Self-consistency prompting*: Builds upon Chain of Thought prompting, using it to create a set of candidate answers, each following a diverse set of reasoning paths to arrive at the final output. These diverse paths are then aggregated by choosing the most consistent answer [76]. Self-consistency relies on the idea that tasks requiring complex reasoning often have multiple paths leading to the correct answer and that we can have more confidence in the final answer when it is arrived at through multiple paths.

(g) *Let's think step by step prompting*: Also builds upon the idea of Chain of Thought prompting (forcing the model to decompose multi-step problems into intermediate steps), but instead of doing this with few-shot prompting, similar behavior is achieved with zero-shot prompting by simply adding "Let's think step by step" to the end of each question [77].

(h) *Self-ask prompting*: Also builds upon Chain of Thought prompting, but unlike CoT, Self-ask clearly demarcates the beginning and end of every sub-question [78]. Since each sub-question is clearly defined, the authors also explore using a search engine to answer each sub-question instead of relying on the model to generate the answer.

(i) *Scratchpad prompting*: Related to Chain of Thought prompting, the model is prompted to break down a task into steps before arriving at a final answer. However, the focus here is on predicting the outcome of computations (like long addition or executing Python programs) [79]. A custom transformer model is prompted to generate the execution trace (the sequence of statements a program executes) line by line before arriving at the final output.

2. ***Automated template learning***: While manual design is common, several approaches have been proposed to automate the template design process for both discrete and continuous prompts.

(a) *For discrete prompts*

- *Prompt mining*: Given a set of training inputs and outputs, this approach identifies frequently occurring words between the input and output. These "middle words" can be used to create a template like "[X] middle words [Z]" [80].
- *Prompt paraphrasing*: An initial prompt is paraphrased into a set of candidate prompts, from which the best performing one is selected. Paraphrasing techniques include back-translation, synonym replacement, or using a language model to generate paraphrases directly [80].
- *Gradient-based search*: Gradient-based search is applied iteratively over actual tokens in a prompt to find sequences that trigger the desired output [81, 82].
- *Prompt generation*: Involves using a language model to generate prompts. Prompt engineering is approached here as a text generation task, and a pre-trained model is fine-tuned on a training set to generate suitable prompts [83].
- *Prompt scoring*: Starting from a set of hand-crafted candidate templates of the form {head, relation, tail}, Davison et al. [84] use an autoregressive language model to score each prompt and select the one with the highest log-likelihood. In this approach, a custom template is created for each input.

(b) *For continuous prompts*

- *Prefix tuning*: A sequence of continuous vectors is added before the input data (prefix matrix). This prefix is then optimized to maximize the log-likelihood of the output [85]. Similar to other prompting methods, the parameters of the language model used to fill the answer slot [Z] remain frozen (what is being fine-tuned is the prompt itself).
- *Tuning initialized with discrete prompts*: A discrete prompt (created either manually or automatically) is used to initialize the continuous prompt, which is then optimized to maximize task accuracy [86]. The research shows that this informed initialization provides a better starting point for the optimization process.
- *Hard-soft prompt hybrid tuning*: In this approach, tunable soft prompts are inserted into hard prompt templates, creating a hybrid prompt composed of both discrete and continuous components [87].

The output from a language model is often a sequence of tokens that may not be directly usable. For instance, when using a language model to answer a question, the output may include words irrelevant to the question (such as long rambling sentences), extraneous tokens (such as punctuation or stop words), or out-of-domain classes when using the model for classification. Therefore, a post-processing step is often necessary to transform the answer slot's output [Z] into the desired final output y. This post-processing step is sometimes referred to as *answer engineering* [49]. In the Text2SQL task, for instance, the language model's output [Z] needs to be mapped into a valid SQL query, which is the final output y.

2-5- Prompting

Having introduced prompting, this section presents the architecture that enables it: the Transformer [5]. This class of neural network excels at learning relationships between sequences of tokens, like natural language questions and their corresponding SQL answers. The Transformer is an instance of an encoder-decoder sequence-to-sequence model. The encoder "reads" the input (question in our example) and transforms it into a comprehensive representation. It achieves this by stacking multiple encoder layers, each processing the input and passing it on to the next layer with a deeper understanding. These layers utilize self-attention, a mechanism allowing the model to consider other parts of the input sequence when encoding a specific token, resulting in a richer representation.

The decoder, conversely, "writes" the output (SQL query in our example) based on the encoder's representation and its own previous outputs. It also utilizes multiple decoder layers, each containing a self-attention mechanism. However, it adds an extra attention layer that focuses on the most relevant parts of the encoder's representation. This allows the decoder to prioritize specific aspects of the input question when generating the output sequence.

The power of the Transformer architecture lies in the attention mechanism, which is a way for the model to learn a mapping between the input and the output in a way that allows it to consider other tokens in the input as it encodes a given token, thereby creating a better encoding for that particular token. Conceptually speaking, this can be thought of in the following way: a word or token by itself has a meaning, but that meaning is deeply affected by its context, which can be any other word (or words) before or after the one in question. While encoder-decoder models (as the one described in general terms above) do exist, the encoder and decoder architectures can be implemented independently, and different combinations have been shown to be more suitable to specific tasks:

- ***Encoder-only models*** (also known as *auto-encoding* Transformer models) such as BERT [36] tend to excel at tasks that require an understanding of the full input, such as sentence classification, named entity recognition and extractive question answering.
- ***Decoder-only models*** (also known as *auto-regressive* Transformer models) such as GPT-3 [9] are best suited to tasks involving text generation.
- ***Encoder-decoder models*** (also known as *sequence-to-sequence* Transformer models) such as T5 [12] are well suited to tasks revolving around generating new sentences depending on a given input, such as summarization, translation, or generative question answering.

The GPT family of models has been fundamental for the recent popularization of LLMs. GPTs are decoder-only architectures. This means they solely consider the left-hand context of a token when generating its embedding, essentially masking the right-hand side. As a consequence, they produce text one token at a time, from left to right, earning them the label "autoregressive models."

One relevant member, GPT-3 [9], consists of 175 billion parameters (the highest at its release). Trained on a massive dataset of web pages, books, Wikipedia entries, and other sources (like Common Crawl https://commoncrawl.org/the-data/), it produced impressive results across various tasks. This included text classification, question answering, summarization, translation, and even 3-digit arithmetic, all in a zero-shot or few-shot setting without specific fine-tuning. It even demonstrated limited success in generating correct and executable code. Building on this, a code-focused version named Codex was fine-tuned on publicly available code from GitHub [40].

GPT-3 was followed by GPT-3.5 [10] (also called InstructGPT) and GPT-4 [11]. GPT-3.5 consists of a supervised fine-tuning of GPT-3 on a large set of labeled data and additional fine-tuning using reinforcement learning from human feedback (RLHF), in which human feedback is used as a reward signal to improve the model further. Despite having only 1.3 billion parameters (100 times fewer than GPT-3), it was shown to outperform GPT-3 on several tasks and is faster and cheaper to run. While details about GPT-4's architecture, hardware, training data, and methods remain undisclosed in its technical report, it differs from its predecessors for its ability to process both text and images (multimodal) and handle larger input contexts. This has led to superior performance on a wide range of academic and professional NLP tasks, including those in non-English languages.

3- Research Methodology

This section provides a detailed description of the methods employed throughout this study. We first discuss the data used, encompassing the Spider benchmark and its Portuguese translation. We then explore the factors influencing the design of prompts used to guide the AI models. Next, the metrics used to assess the performance of the models are explained. Finally, we detail the configuration of the experiment, including data preparation and model evaluation procedures.

The first step consists of preprocessing the original Spider dataset. This involves extracting database schemas and sample content into a separate JSON file for easier use. This preprocessed data is then combined with the Portuguese translation of the Spider questions to create the final dataset for the experiment. With the data prepared, prompts are constructed for each question. These prompts combine the Portuguese question with relevant information, including database schema details and sample content when applicable. We utilize these prompts and specific model parameters to make calls to the OpenAI API for each question. Finally, the API responses are processed to extract the predicted SQL queries.

Once all questions in the development set have predicted queries, we employ the official Spider evaluation script (described in the "Evaluation Metrics" section) to compute performance metrics. Figure 1 provides a visual representation of the methodology.

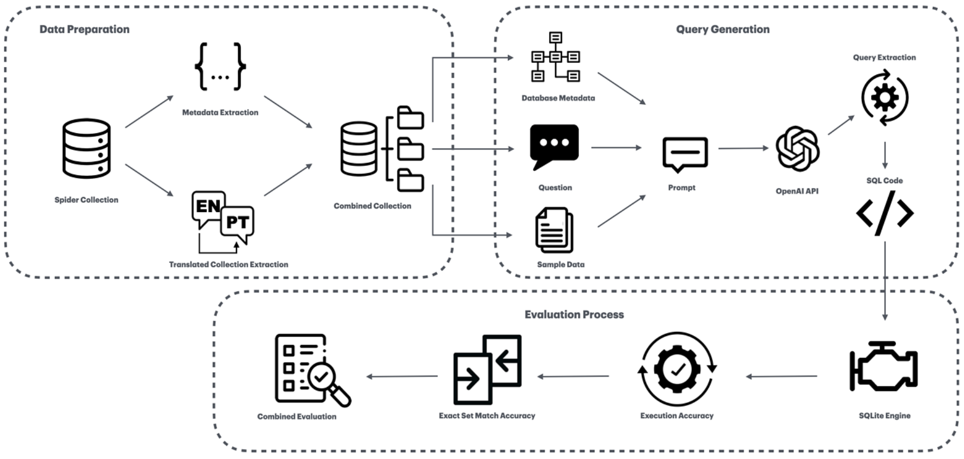

Figure 1. Overview of the methodology

3-1- Dataset

While numerous benchmarks exist for Text2SQL tasks, research on low-resource languages like Portuguese lags behind that on English. Currently, the most comprehensive Portuguese Text2SQL benchmark is the translation of the Spider dataset by José & Cozman [16].

Spider is a large and comprehensive dataset for semantic parsing and Text2SQL tasks. It includes 10,181 questions in English and 5,693 unique corresponding SQL queries spanning four difficulty levels: from *easy* to *extra hard*. Moreover, it is a cross-domain dataset with 200 distinct databases, each containing multiple tables, covering 138 domains. The 200 databases were collected from distinct sources: college database courses and textbooks, online SQL tutorials, DatabaseAnswers (the original URL cited by the authors as http://www.databaseanswers.org/ is no longer available, but a copy of the website is accessible through the Internet Archive: https://web.archive.org/web/ 20210303213924/http://databaseanswers.org/), and WikiSQL [21]. These were then manually treated by correcting or adding schemas and populating the empty tables with sample data (although after gaining access to the data, we found that some databases did have empty tables and only their schemas were available).

The databases were then manually annotated with English-language questions that could be answered by querying the database and the corresponding SQL queries to answer them. The questions were written in such a way as to be diverse, natural, and reflective of how databases are used in practice. For each database, the SQL queries cover all the following components: SELECT with multiple columns and aggregations, WHERE, GROUP BY, HAVING, ORDER BY, LIMIT, JOIN, INTERSECT, UNION, NOT IN, OR, AND, EXISTS, LIKE, and nested queries [17]. It is important to note that the dataset includes no ambiguous questions or questions that require common sense knowledge beyond the scope of the database to answer.

As mentioned earlier, each SQL query is annotated with a difficulty level based on its complexity. This classification considers the number and type of components present, such as clauses, nested subqueries, aggregations, and joins. More elements generally indicate a more complex query. Additionally, the dataset creators divided the 200 databases into training, development, and testing sets. These sets contain roughly 77%, 8%, and 15% of the databases, respectively. Importantly, questions related to the same database are guaranteed to appear only in the same set. This ensures that model evaluations reflect their ability to generalize to unseen databases, a crucial skill. It is important to note that the test set is not publicly available and is used solely for official model evaluation.

The data are available in the form of SQLite databases, with the schema and sample data in the form of SQL files. The Spider databases, questions, queries, and official evaluation script can be found through the project's web page (*https://yale-lily.github.io/spider*) and consist of the following files:

1. ***For each database:***
   - schema.sql: The file to create the database, with the schema and sample data.
   - database_name.sqlite: The SQLite database itself.
2. ***Questions and queries:***
   - train_spider.json: The training set of questions, databases they refer to, and the corresponding SQL queries.
   - train_others.json: An additional (much smaller) split of the training set.
   - dev.json: The development set of questions, databases they refer to, and the corresponding SQL queries.
3. ***evaluation.py:*** The official evaluation script

3.1.1. Factors Affecting Prompting Approaches

In José & Cozman [16], as part of their work on adapting the RAT-SQL+GAP system [46] to the Portuguese language, the authors also produce a Portuguese translation of the Spider benchmark. This translation focused solely on the natural language questions, leaving the SQL queries and databases in their original English format. This translation was done via the Google Cloud Translation API (https://googleapis.dev/python/ translation/latest/index.html) and then manually revised by the authors, resulting in three new JSON files (one for each split). These translated datasets are available in the authors' GitHub repository (https://github.com/C4AI/gap-text2sql).

Figure 2 shows examples of the original English questions and their corresponding Portuguese translations. Some of the translations are straightforward, such as the first two. However, the third and fourth examples illustrate some of the challenges of this task: When referencing a particular table column or value, should those be translated as well? Note that in the third question, the values "Treasury" and "Homeland Security" were translated into Portuguese (even though they appear in the table in English), but in the fourth question, the value "The Phantom of the Opera" was not.

QUESTION 1

English: "List the name, born state and age of the heads of departments ordered by age."

Portuguese: "Liste o nome, estado de nascimento e idade dos chefes de departamentos, ordenados por idade."

QUESTION 2

English: "What are the ids of all students for courses and what are the names of those courses?"

Portuguese: "Quais são os ids de todos os alunos para cursos e quais são os nomes desses cursos?"

QUESTION 3

English: "List the states where both the secretary of 'Treasury' department and the secretary of 'Homeland Security' were born."

Portuguese: "Liste os estados onde nasceram o secretário do departamento de 'Tesouro' e o secretário de 'Segurança Interna'."

QUESTION 4

English: "Show names of actors that have appeared in musical with name 'The Phantom of the Opera'."

Portuguese: "Mostre nomes de atores que apareceram no musical com o nome 'The Phantom of the Opera'."

Figure 2. Examples of questions translated from the original Spider dataset to Portuguese

3.1.2. Evaluation Metrics

The authors of the Spider benchmark reference three evaluation metrics: *Component Match*, *Exact Set Match*, and *Execution Accuracy*. An official evaluation script is also provided, which can be used to compute these. The script takes as input two text files: one with the generated SQL queries and another with the gold SQL queries.

Component Match (CM) is the average exact match between the predicted and ground truth SQL queries for different SQL components. Each of the following components is decomposed into sets of subcomponents, and the predicted and gold query sets are compared, checking if they match exactly:

- SELECT
- WHERE
- GROUP BY
- ORDER BY
- KEYWORDS: all SQL keywords without column names and operators

For example, to evaluate the SELECT component in a predicted query such as SELECT avg(col1), max(col2), min(col1), it is first decomposed into the following set: (avg, min, col1), (max, col2). The same is done for the gold query, and the two sets are compared to check if they match exactly. A model's overall performance on each component is computed as the F1 score on exact set matching.

Exact Set Match (ESM) measures whether the predicted query as a whole is equivalent to the gold query. Each component of the predicted query is evaluated as described in the previous section on CM, and the query is considered correct only if all of the components match that of the gold query. ESM can be reported both with or without values, the difference being that the former requires the predicted query to contain the same values (such as in the WHERE clause) as the gold query, while the latter does not. For example, if the gold query is SELECT * FROM table WHERE col1 = 1, then the predicted query SELECT * FROM table WHERE col1 = 2 would be considered 100% correct if ESM is reported without values, but not so if ESM is reported with values. ESM without values is a weaker metric than ESM with values, and both are commonly reported in the literature.

There is often more than one way of correctly translating a question into a SQL query (see Figure 3), and because CM and ESM rely on a gold query to evaluate whether a predicted query is correct, they are susceptible to false negatives when the predicted query is semantically equivalent to the gold query but syntactically different. To address this, Execution Accuracy (EA) measures whether the predicted and gold queries return the same result when executed against the database. Note that EA can also produce false positives when two semantically different queries return the same result. This is particularly problematic for simpler questions and databases, where the same result answers many different questions.

```
-- QUERY 1 SELECT
   s.id, c.name
FROM
   students s JOIN
   courses c
      ON s.course_id = c.id;

-- QUERY 2 SELECT
   s.id, c.name
FROM
   students s,
   courses c
WHERE s.course_id = c.id;
```

Figure 3. Queries 1 and 2 are semantically equivalent but syntactically different. While Query 1 uses an explicit join, Query 2 uses an implicit join. Both queries evaluate to the same result using EA but are evaluated differently in CM and ESM

3-2- Experimental Setup

This investigation focuses on the use of the GPT3.5 [10] and GPT-4 [11] models, accessible through the OpenAI API (https://platform.openai.com/docs/). We were also originally interested in using the code-specific Codex models [40], but API access to these was discontinued while writing this paper.

To assess how effectively these models translate Portuguese natural language questions into SQL queries, we employed them to answer the Portuguese questions within the Spider benchmark's development set. We then compared the predicted queries to the reference (gold) queries and utilized the official evaluation script to calculate the metrics described earlier. The development set was chosen over the test set because the latter is not publicly available for independent evaluation.

We investigate the effect that prompt engineering has on the quality of the outputs by comparing the results of different zero-shot prompts. The prompt structures include the following characteristics:

- ***Question***: The Portuguese question to be translated into SQL.
- ***Database schema***: The schema of the database that the question references, including table and column names, column types, and foreign key relationships.
- ***Database content***: A sample of the database's content. We experimented with different sample sizes, including 1 and 3 rows.

The prompts used in the experiments include the following four styles:

1. ***Question*** prompt: The question is presented as a standalone prompt, and no additional database information is provided. Example: *"Show all artists from the movie 'Standalone'."*
2. **+ *Schema*** prompt: The database schema is also provided in addition to the question. In this case, a dictionary is added to the intial question prompt as a string, reflecting the database metadata. Example: *"{ 'database_1': { 'tables': { 'table_1': { 'cols': { 'column_1': { 'type': 'column_type',}…"*
3. **+ 1 *Example*** prompt: In addition to the question and schema, the prompt includes a single random row of database content. This row is included in the prompt as an extended dictionary of the schema prompt. Example: *"{'database_1': { 'tables': { 'table_1': { 'cols': { 'column_1': { 'type': 'column_type',* ***'sample': ['example1']****} "*
4. **+ 3 *Examples*** prompt: In addition to the question and schema, the prompt includes three random rows of database content. The logic for this prompt is the same as presented in the example above, but three examples are provided.

The number of examples included in the prompts was capped at three for the following reasons:

- ***Cost***: Each call to the OpenAI API incurs a cost (at the time of writing, the cost is $0.03/1k tokens for the gpt-4 family of models with 8k token context, and $0.002/1k tokens for the gpt-3.5-turbo family of models) which, given the prompt size and number of calls required for the experiments, would make it untenable to test more prompt styles.
- ***Context length***: While the GPT-4 models support up to 8,192 tokens of context, the GPT-3.5 models only support up to 4,096 tokens. Given the structure of the prompts (which include the question, schema, examples, and additional

instructions), in practice, this means that the maximum number of examples that could be included in a prompt is seven.

- ***Performance***: Using the Codex model, Rajkumar et al. [8] found that more database content in the prompt can harm performance, with execution accuracy peaking at three rows of sample data before significantly decreasing in performance as more rows are added.

4- Results and Discussion

Table 2 summarizes the performance of GPT-3.5 and GPT-4 models on the Spider development set, using various prompt styles as described earlier. These results were obtained using the official Spider evaluation script.

Table 2. Spider benchmark. Development set performance across different prompt styles for the GPT-3.5 and GPT-4 models, as measured by Execution Accuracy (EA) and Exact Set Match (ESM, with and without values)

Prompt	EA		ESM (without values)		ESM (with values)	
	GPT-3.5	GPT-4	GPT-3.5	GPT-4	GPT-3.5	GPT-4
Question	0.030	0.044	0.033	0.044	0.030	0.041
+ Schema	0.355	0.428	0.302	0.379	0.266	0.308
+ 1 Example	0.371	0.450	0.301	0.386	0.279	0.334
+ 3 Examples	0.391	0.467	0.310	0.394	0.286	0.349

As expected, GPT-4 consistently outperforms GPT-3.5 across all metrics. On average, GPT-4 achieves a 27.0% (6.1 percentage points) improvement in EA and a 28.5% (6.4 percentage points) improvement in ESM. Additionally, prompts that provide more context about the data being queried, such as schema information and examples, lead to better performance.

Table 2 reveals that the best results are achieved by GPT-4 with a prompt that includes both schema information and three examples. This configuration achieves an EA score of 0.467, an ESM score (excluding values) of 0.394, and an ESM score (including values) of 0.349.

However, as discussed in Section "Error Analysis", it should be noted that the above results underestimate the performance of the approaches tested here. This is due to the evaluation metrics used by Spider and the errors in the gold queries we observed. Therefore, the results presented in Table 2 should be viewed as a conservative estimate of GPT-3.5 and GPT-4's capabilities in Portuguese Text2SQL tasks.

4-1-Comparison with Previous Works

The only previous work reporting results on this Portuguese translation of the Spider dataset is José & Cozman [16], where the RAT-SQL+GAP system [46] was adapted to a multilingual context. However, the authors only report results for ESM without values. Their best-performing model achieved an ESM (without values) of 0.595, significantly better than the best-performing model here (+51.0% or 20.1 pp higher). This was accomplished by fine-tuning a multilingual BART model [48] on a dataset consisting of both Portuguese and English questions.

This substantial performance difference might be attributed to the fine-tuning process employed in their approach. Fine-tuning potentially allows their predicted queries' syntax to closely resemble the reference queries in Spider. As mentioned in the "Error Analysis" section, the models in this study often generate queries with correct semantics that answer the questions accurately. However, due to the evaluation metrics, these queries are marked as incorrect. Since the zero-shot approach used here does not involve fine-tuning, it is more challenging to influence the query style. Results for the fine-tuned GPT-3 models presented by Rajkumar et al. [8] and included in Table 4 (discussed later) seem to support this hypothesis (Table 3).

Table 3. Comparison of the best performing models of this paper and previous works on the Spider dataset

Model & approach	Spider dataset language	Best performance		Reference
		ESM	EA	
GPT-4 with schema + 3 example prompt	Portuguese	0.394	0.467	This paper
mRAT-SQL+GAP	Portuguese	0.595	-	José & Cozman [16]
Codex-davinci with schema + 3 example prompt	English	-	0.670	Rajkumar et al. [8]

Table 4. Comparison of EA for unmodified and fine-tuned GPT-3 models using schema-style prompts, as reported by Rajkumar et al. [8]

GPT-3 model	EA		EA increase
	Unmodified	Fine-tuned	
Ada	0.023	0.202	8.8x
Babbage	0.057	0.348	6.1x
Curie	0.126	0.513	4.1x

Rajkumar et al. [8] report results for the original Spider dataset in English, using zero-shot approaches similar to the ones tested here with a variety of models, including various sizes of GPT-3 (although OpenAI does not make openly available the parameter counts for their models, in their comparison to the results published in [9, 88] they found that "Ada, Babbage, Curie, and Davinci line up closely with 350M, 1.3B, 6.7B, and 175B, respectively"), both unmodified and fine-tuned, and Codex. Their most successful approach utilized the fine-tuned Codex-Davinci model with a prompt including schema and three examples (matching our best-performing configuration). This achieved an EA score of 0.670, which is 43.5% (or 20.3 percentage points) higher than our best model. They do not report ESM values.

As mentioned earlier, the authors also present EA results for fine-tuned GPT-3 models (Ada, Babbage, and Curie) using the same schema-style prompt. Such results (reported in Table 4) show the remarkable impact of fine-tuning on the performance of these models. Fine-tuning yielded improvements ranging from 4.1x for the larger Curie model to an 8.8x for the smaller Ada model. Their unmodified GPT-3-Davinci model with a schema-style prompt achieves an EA of 0.263, which is lower than the results presented here for the GPT-3.5 and GPT-4 models with the same prompt style (0.355 and 0.428, respectively). This substantial difference in performance between Codex-Davinci and a similarly sized GPT-3 model further emphasizes the benefits of fine-tuning for Text2SQL tasks; the primary distinction between these two models lies in the fine-tuning process that Codex undergoes.

4-2- Error Analysis

In this section, we analyze the errors made by our best-performing approach (GPT-4 with schema + 3 example prompt) in more detail. Table 5 provides a summary of these errors, while Table 7 shows examples of the various error types.

Table 5. Overview of manual error analysis on 100 randomly selected incorrect queries generated by the GPT-4 model with schema + 3 example prompt. The number of errors in each category is shown in the rightmost column

Error type	#
Ambiguous correct	**56**
—SELECT extra columns	7
—SELECT convention	17
—Argmax	2
—Other	30
Semantic incorrect	**28**
—Shortcuts	2
—GROUP BY convention	2
—Other	24
Invalid SQL	**9**
—Ambiguous column	0
—No such column or table	9
Gold incorrect	**7**

One hundred random output queries marked as incorrect by Spider's official evaluation script were manually analyzed and classified into error categories similar to those used by Rajkumar et al. [8], described below.

Answers classified as ***Ambiguous correct*** are when the predicted query is semantically different from the gold query but still correctly answers the question posed, causing it to be inaccurately marked as incorrect by the Spider evaluation script. These are further broken down into:

- SELECT extra columns: When GPT-4 includes additional columns in the SELECT clause that are not present in the gold query.

- SELECT convention: When GPT-4 selects a different but semantically equivalent set of columns than the gold query (e.g., name instead of ID), or when it adds an alias to a column (e.g., count(*) as count instead of count(*)), causing the official evaluation script to mark it as incorrect.
- Argmax: When the predicted and gold queries differ in how a min/max is resolved in the case of a tie.
- Other. This subcategory aggregates many different types of errors, such as:
  - Use of explicit joins instead of implicit ones, or vice versa.
  - Use of subqueries instead of joins, or vice versa.
  - Including unnecessary joins in the query and unnecessary fields in the GROUP BY clause, which do not affect the result or meaning of the query.
  - When asked for the *most* or *least* of something, answer the questions by ordering the results and limiting them to the first element instead of filtering by the maximum or minimum value (e.g., order by count(*) desc limit 1 instead of where count(*) = (select max(count(*)) from table).

When both the Spider evaluation script and our manual annotation agree that the predicted query is incorrect, they are classified as ***Semantically incorrect***, with the subcategories:

- Shortcuts: When GPT-4 uses particular table values or "world knowledge" acquired from its pretraining instead of employing the precise literals present in the original question, as done in the gold query.
- GROUP BY convention: When a non-primary key (such as a name or title field) is incorrectly used to group the results of an aggregate function.
- Other.

Invalid SQL is when the predicted query is invalid SQL code:

- Ambiguous column: When the predicted query contains an ambiguous column name (i.e., it exists in more than one table in the database).
- No such column or table: When the predicted query contains a column or table name that does not exist in the database. This usually happens when GPT-4 includes the database name in front of the table name (e.g., database.table instead of just table), causing it to fail the official evaluation script.

Finally, when performing this manual analysis, we found several cases in which the gold query incorrectly answered the question posed, but the predicted query was correct, leading them to be marked incorrect. These are classified as **Gold incorrect**. Examples of these incorrect gold queries are presented in Table 6.

Table 6. Examples of incorrect gold queries in Spider's development set

Question in English	List the name, the date, and the result of each battle.
Gold query	SELECT name, date FROM battle
Predicted query	SELECT name, date, result FROM battle
Comment	The question also asks for the result of each battle, which is not present in the gold query.
Question	Liste o sobrenome do dono do cão mais novo.
Question in English	List the surname of the owner of the youngest dog.
Gold query	T2 ON T1.owner_id = T2.owner_id WHERE T2.age = (SELECT Max(age) FROM dogs)
Predicted query	SELECT owners.last_name FROM owners JOIN dogs ON owners.owner_id = dogs.owner_id WHERE dogs.date_of_birth = (SELECT Max(date_of_birth) FROM dogs)
Comment	To determine the youngest dog, the gold query filters the results by the dog with the maximum age (which actually corresponds to the oldest dog). The predicted query correctly filters by the maximum date of birth.
Question	Liste o nome dos professores cuja cidade natal não seja "Little Lever Urban District"
Question in English	List the name of the professors whose hometown is "Little Lever Urban District".
Gold query	SELECT name FROM teacher WHERE hometown!= "little lever urban district"
Predicted query	SELECT name FROM teacher WHERE hometown <> 'Little Lever Urban District'
Comment	The gold query lowercases the string "Little Lever Urban District" in the WHERE clause, which causes it to not match the string in the database. The predicted query correctly keeps the same casing as presented in the question.

In this manual analysis, similar to the one by Rajkumar et al. [8], the other categories in Ambiguous correct and Semantic incorrect were the most frequent. This suggests these categories could benefit from further breakdown. However, we maintained the same categories as the previous study to facilitate comparisons.

In addition to the error types mentioned earlier, we encountered instances where GPT-4 responded to questions related to the "most" or "least" of something with a complete ordered list, thus failing to filter it to only the first or last element. While this is technically incorrect and was marked as such in both the evaluation script and the manual annotation, a human looking at the results would be able to answer the question correctly.

Table 5 shows that 56% of the answers marked as incorrect by Spider's official evaluation script can be classified as *ambiguous correct*, meaning that although they differ in some way from the gold query, they still correctly answer the respective questions posed. This manual analysis highlights one of the main limitations of automatic evaluation metrics for Text2SQL, such as the ones used in Spider, which in this case, significantly underestimate the performance of our approaches. In addition, the 7% of gold queries we found to be incorrect in the manual analysis further strengthens the argument that the results in Table 2 underestimate the performance of the models under consideration.

4-3- Dataset Memorization

The popularity of the Spider benchmark raises a potential concern: since its development set is publicly available, could GPT-4 have simply memorized the answers during training instead of learning how to answer the questions? We believe this is unlikely for several reasons. However, we do not believe this to be the case.

First, we used the Portuguese translation of Spider's questions, a less common version compared to the widely available English one. Second, the significant performance jump (roughly 10x) observed when switching from the "Question" prompt to the "Schema + 3 example" prompt suggests GPT-4 is not just memorizing answers. If memorization were dominant, the simpler "Question" prompt would be expected to perform better. Third, as discussed in Section "Error Analysis", GPT-4's predictions are often *Ambiguous correct*, meaning that they differ from the gold query but are still semantically correct (that is, GPT-4 has correctly answered the question using a different approach than the one used in the gold query). It should also be noted that GPT-4 consistently adopts a different aliasing convention than the one used in the gold queries, tending to pick meaningful aliases for tables and columns instead of the generic style used in the gold queries (e.g., T1 and T2). Finally, research on memorization in GitHub Copilot (a product based on the Codex model) by Ziegler [89] provides further evidence. While Copilot can quote publicly available code verbatim under specific prompting, this happens rarely and typically involves widely used code snippets, often at the beginning of a file. This suggests memorization is not a common behavior in these models.

While the proposed approach does not achieve state-of-the-art performance, the results are promising. The models, particularly GPT-4, can generate adequate SQL queries for answering Portuguese questions across various databases. As shown in Table 2, incorporating schema information and database content within the prompts is crucial to achieving satisfactory results.

Although we have tested the efficacy of several zero-shot prompts, it is possible to investigate the use of few-shot prompts and multi-step approaches. In these multi-step approaches, the model generates candidate queries that are subsequently executed against the database. If necessary, these candidate queries can be refined before presenting them to the user. We have also pointed out several issues with the automatic evaluation process used in Spider, which we believe significantly underestimates the performance of the approaches tested here and their applicability in real-world scenarios. More robust evaluation processes could be explored.

Additionally, relying on a translated benchmark might limit the testing of the model's ability to handle the complexities of real-world Portuguese queries. These complexities include regional language variations, colloquialisms, and intricate SQL constructs not captured by the benchmark. In particular, the Spider dataset excludes ambiguous or poorly defined questions, which are common in real-world scenarios. Future work could explore the models' efficacy in a more conversational setting where follow-up questions can be used for clarification. Developing additional Portuguese Text2SQL benchmarks could also be a valuable avenue for future research.

Finally, comparing our results with those achieved using fine-tuned models (both in Portuguese [16] and English [8]) shows the significant performance gains associated with fine-tuning for the Text2SQL task. This is likely particularly true for benchmarks like Spider, where automatic evaluation focuses on query structure rather than the ability to answer the question. Fine-tuned models can better adapt to the specific evaluation style, leading to better performance on that benchmark.

Table 7. Examples of some of the most common error types made by the GPT-4 model with schema+ 3 example prompt

Ambiguous correct	**SELECT extra columns**
Question	Liste o episódio de todas as séries de TV classificadas por classificação.
Question in English	List episodes of all TV series sorted by rating.
Gold query	SELECT episode FROM tv_series ORDER BY rating
Predicted query	SELECT episode, rating FROM tv_series ORDER BY rating ASC
Comment	The predicted query includes the rating column in the SELECT clause, which provides additional information in answering the question but is not present in the gold query.
Ambiguous correct	**SELECT convention**
Question	Qual é o número de funcionários de cada cidade?
Question in English	What is the number of employees in each city?
Gold query	SELECT Count(*), city FROM employee GROUP BY city
Predicted query	SELECT city, Count (employee_id) AS num_funcionarios FROM employee GROUP BY city
Comment	The predicted query aggregates the results by counting the employee_id column instead of using the * wildcard.
Ambiguous correct	**Other**
Question	Encontrar a id e o nome do museu com mais funcionários
Question in English	Find the ID and name of the museum with the most employees
Gold query	SELECT museum_id, name FROM museum ORDER BY num_of_staff DESC LIMIT 1
Predicted query	SELECT museum_id, NAME FROM museum WHERE num_of_staff = (SELECT Max(num_of_staff) FROM museum)
Comment	The predicted query uses a subquery to find the maximum number of staff, while the gold query orders the results by the number of staff in descending order and limits the results to the first row.
Semantic incorrect	**Other**
Question	Qual é o modelo do carro com peso menor que a média?
Question in English	What car model weighs less than average?
Gold query	SELECT T1.model FROM car_names AS T1 JOIN cars_data AS T2 ON T1.makeid = T2.id WHERE T2.weight < (SELECT Avg(weight) FROM cars_data)
Predicted query	SELECT model_list.model FROM model_list JOIN cars_data ON model_list.modelid = cars_data.id WHERE cars_data.weight < (SELECT Avg(weight) FROM cars_data)
Comment	The predicted query erroneously uses the model_list table instead of the car_names table.
Invalid SQL	**No such column or table**
Question	Quantas companhias aéreas existem?
Question in English	How many airlines exist?
Gold query	SELECT count(*) FROM AIRLINES
Predicted query	SELECT COUNT(uid) FROM flight_2.airlines
Comment	GPT-4 includes the database name in front of the table name (which would be correct in many SQL dialects), but the Spider evaluation script recognizes this as a different table, marking it incorrect

5- Conclusion

Traditionally, Text2SQL research has centered on English, creating a barrier for users who primarily interact with information in other languages. This work explored the use of the GPT family of models for the Text2SQL task in the Portuguese language. In particular, we investigated the use of the GPT-3.5 and GPT-4 models under a zero-shot setting on a Portuguese translation of the popular Spider benchmark. While the results suggested that there is room for further improvement, the findings of this research demonstrated the potential of LLMs in this domain with different prompting strategies. As this approach is further refined, we expect it will make database interaction ever more accessible to non-technical Portuguese speakers and possibly other low-resource languages, opening up new possibilities for the use of databases in a wide range of applications.

All in all, this research advanced the knowledge of Text2SQL and LLMs for low-resource languages. By demonstrating the effectiveness of the proposed approach, this study paved the way for more research focused on other low-resource languages. This can allow users who may not be fluent in English or lack technical knowledge of query languages to interact with databases in their native tongue. This democratization of Text2SQL has the potential to provide better information access and retrieval to a wider range of users, fostering a more inclusive use of LLMs for Text2SQL.

Future work may consider exploring the impact of different database schema complexities on LLM performance using zero-shot prompting and an analysis of the potential of combining zero-shot prompting with other techniques like question reformulation or summarization to enhance LLM understanding in Text2SQL tasks.

6- Declarations

6-1- Author Contributions

Conceptualization, M.J., Y.P., and M.C.; methodology, M.J. and Y.P.; software, M.J.; validation, Y.P., F.P., and M.C.; formal analysis, M.J.; investigation, M.J.; resources, M.C. and A.P.; data curation, M.J. and Y.P.; writing—original draft preparation, M.J. and Y.P.; writing—review and editing, Y.P., F.P., M.C., and A.P.; visualization, M.J. and Y.P.; supervision, M.C. and F.P.; project administration, M.C. and A.P.; funding acquisition, M.C. and A.P. All authors have read and agreed to the published version of the manuscript.

6-2- Data Availability Statement

Publicly available datasets were analyzed in this study. This data can be found here: https://yale-lily.github.io/spider.

6-3- Funding

The authors received no financial support for the research, authorship, and/or publication of this article.

6-4- Institutional Review Board Statement

Not applicable.

6-5- Informed Consent Statement

Not applicable.

6-6- Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this manuscript. In addition, the ethical issues, including plagiarism, informed consent, misconduct, data fabrication and/or falsification, double publication and/or submission, and redundancies have been completely observed by the authors.

7- References

[1] Price, P. J. (1990). Evaluation of spoken language systems. Proceedings of the Workshop on Speech and Natural Language - HLT '90, 91–95. doi:10.3115/116580.116612.

[2] Zelle, J. M., & Mooney, R. J. (1996). Learning to parse database queries using inductive logic programming. Proceedings of the National Conference on Artificial Intelligence, 4-8 August, 1996, Portland, United States.

[3] Li, F., & Jagadish, H. V. (2014). Constructing an interactive natural language interface for relational databases. Proceedings of the VLDB Endowment, 8(1), 73–84. doi:10.14778/2735461.2735468.

[4] Saha, D., Floratou, A., Sankaranarayanan, K., Minhas, U. F., Mittal, A. R., & Özcan, F. (2016). ATHENA: An ontologydriven system for natural language querying over relational data stores. Proceedings of the VLDB Endowment, 9(12), 1209–1220. doi:10.14778/2994509.2994536.

[5] Vaswani, A. (2017). Attention is all you need. Advances in Neural Information Processing Systems. 31st Conference on Neural Information Processing Systems, 4-9 December, 2017, Long Beach, United States.

[6] Wang, B., Shin, R., Liu, X., Polozov, O., & Richardson, M. (2020). RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 1-12. doi:10.18653/v1/2020.acl-main.677.

[7] Scholak, T., Schucher, N., & Bahdanau, D. (2021). PICARD: Parsing Incrementally for Constrained Auto-Regressive Decoding from Language Models. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 1-7. doi:10.18653/v1/2021.emnlp-main.779.

[8] Rajkumar, N., Li, R., & Bahdanau, D. (2022). Evaluating the Text-to-SQL Capabilities of Large Language Models. arXiv Preprint, arXiv:2204.00498. doi:10.48550/arXiv.2204.00498.

[9] Brown, T. B. (2020). Language models are few-shot learners. arXiv Preprint, arXiv:2005.14165.doi:10.48550/arXiv.2005.14165.

[10] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C., Mishkin, P., ... & Lowe, R. (2022). Training language models to follow instructions with human feedback. Advances in neural information processing systems, MIT Press, Cambridge, United States. doi:10.48550/arXiv.2203.02155.

[11] Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F. L., ... & McGrew, B. (2023). Gpt-4 technical report. arXiv preprint arXiv:2303.08774. doi:10.48550/arXiv.2303.08774.

[12] Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., ... & Liu, P. J. (2020). Exploring the limits of transfer learning with a unified text-to-text transformer. Journal of Machine Learning Research, 21(140), 1-67.

[13] Scao, T. L., Fan, A., Akiki, C., Pavlick, E., Ilić, S., ... & Bari, M. S. (2022). Bloom: A 176b-parameter open-access multilingual language model. arXiv Preprint, arXiv:2211.05100. doi:10.48550/arXiv.2211.05100.

[14] Sun, Y., Wang, S., Feng, S., Ding, S., Pang, C., Shang, J., ... & Wang, H. (2021). Ernie 3.0: Large-scale knowledge enhanced pre-training for language understanding and generation. arXiv preprint arXiv:2107.02137.. doi:10.48550/arXiv.2107.02137.

[15] da Silva, C.F.M., Jindal, R. (2021). SQL Query from Portuguese Language Using Natural Language Processing. In: Garg, D., Wong, K., Sarangapani, J., Gupta, S.K. (eds) Advanced Computing. IACC 2020. Communications in Computer and Information Science. Springer, Singapore. doi:10.1007/978-981-16-0401-0_25.

[16] José, M. A., & Cozman, F. G. (2021). mRAT-SQL+GAP: A Portuguese Text-to-SQL Transformer. In: Britto, A., Valdivia Delgado, K. (eds) Intelligent Systems. BRACIS 2021. Lecture Notes in Computer Science. Springer, Cham, Switzerland. doi:10.1007/978-3-030-91699-2_35.

[17] Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., ... & Radev, D. (2018). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. arXiv preprint, arXiv:1809.08887.. doi:10.48550/arXiv.1809.08887.

[18] Jin, N., Siebert, J., Li, D., Chen, Q. (2022). A Survey on Table Question Answering: Recent Advances. Knowledge Graph and Semantic Computing: Knowledge Graph Empowers the Digital Economy. Communications in Computer and Information Science. Springer, Singapore. doi:10.1007/978-981-19-7596-7_14.

[19] Pasupat, P., & Liang, P. (2015). Compositional Semantic Parsing on Semi-Structured Tables. Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing. doi:10.3115/v1/p15-1142.

[20] Misra, D., Chang, M.-W., He, X., & Yih, W. (2018). Policy Shaping and Generalized Update Equations for Semantic Parsing from Denotations. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing. doi:10.18653/v1/d18-1266.

[21] Zhong, V., Xiong, C., & Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. doi:10.48550/arXiv.1709.00103.

[22] Sun, Y., Tang, D., Duan, N., Ji, J., Cao, G., Feng, X., Qin, B., Liu, T., & Zhou, M. (2018). Semantic Parsing with Syntax- and Table-Aware SQL Generation. Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics. doi:10.18653/v1/p18-1034.

[23] Nan, L., Hsieh, C., Mao, Z., Lin, X. V., Verma, N., Zhang, R., Kryściński, W., Schoelkopf, H., Kong, R., Tang, X., Mutuma, M., Rosand, B., Trindade, I., Bandaru, R., Cunningham, J., Xiong, C., Radev, D., & Radev, D. (2022). FeTaQA: Free-form Table Question Answering. Transactions of the Association for Computational Linguistics, 10, 35–49. doi:10.1162/tacl_a_00446.

[24] Mueller, T., Piccinno, F., Shaw, P., Nicosia, M., & Altun, Y. (2019). Answering Conversational Questions on Structured Data without Logical Forms. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 5901–5909. doi:10.18653/v1/d19-1603.

[25] Herzig, J., Nowak, P. K., Müller, T., Piccinno, F., & Eisenschlos, J. (2020). TaPas: Weakly Supervised Table Parsing via Pre-training. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.398.

[26] Zhu, F., Lei, W., Huang, Y., Wang, C., Zhang, S., Lv, J., Feng, F., & Chua, T.-S. (2021). TAT-QA: A Question Answering Benchmark on a Hybrid of Tabular and Textual Content in Finance. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing. doi:10.18653/v1/2021.acl-long.254.

[27] Sun, H., Ma, H., He, X., Yih, W., Su, Y., & Yan, X. (2016). Table Cell Search for Question Answering. Proceedings of the 25th International Conference on World Wide Web, 771 - 782. doi:10.1145/2872427.2883080.

[28] Glass, M., Canim, M., Gliozzo, A., Chemmengath, S., Kumar, V., Chakravarti, R., Sil, A., Pan, F., Bharadwaj, S., & Fauceglia, N. R. (2021). Capturing Row and Column Semantics in Transformer Based Question Answering over Tables. Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies. doi:10.18653/v1/2021.naacl-main.96.

[29] Chen, W., Chang, M. W., Schlinger, E., Wang, W., & Cohen, W. W. (2020). Open question answering over tables and text. arXiv, preprint arXiv:2010.10439.. doi:10.48550/arXiv.2010.10439.

[30] Baik, C., Jagadish, H. V., & Li, Y. (2019). Bridging the Semantic Gap with SQL Query Logs in Natural Language Interfaces to Databases. 2019 IEEE 35th International Conference on Data Engineering (ICDE), abs 909 1765, 374–385. doi:10.1109/icde.2019.00041.

[31] Xu, X., Liu, C., & Song, D. (2017). Sqlnet: Generating structured queries from natural language without reinforcement learning. arXiv preprint arXiv:1711.04436. doi:10.48550/arXiv.1711.04436.

[32] Yu, T., Li, Z., Zhang, Z., Zhang, R., & Radev, D. (2018). Typesql: Knowledge-based type-aware neural text-to-SQL generation. arXiv preprint, arXiv:1804.09769. doi:10.48550/arXiv.1804.09769.

[33] Iyer, S., Konstas, I., Cheung, A., Krishnamurthy, J., & Zettlemoyer, L. (2017). Learning a neural semantic parser from user feedback. arXiv preprint, arXiv:1704.08760. doi:10.48550/arXiv.1704.08760.

[34] Gür, I., Yavuz, S., Su, Y., & Yan, X. (2018, July). Dialsql: Dialogue based structured query generation. In Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, 1339-1349. doi:10.18653/v1/p18-1124.

[35] Bogin, B., Berant, J., & Gardner, M. (2019). Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics. doi:10.18653/v1/p19-1448.

[36] Devlin, J. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding. arXiv preprint, arXiv:1810.04805. doi:10.48550/arXiv.1810.04805.

[37] Yin, P., & Neubig, G. (2018). TRANX: A Transition-based Neural Abstract Syntax Parser for Semantic Parsing and Code Generation. Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations, 7–12. doi:10.18653/v1/d18-2002.

[38] Lin, K., Bogin, B., Neumann, M., Berant, J., & Gardner, M. (2019). Grammar-based neural text-to-sql generation. arXiv preprint, arXiv:1905.13326. doi:10.48550/arXiv.1905.13326.

[39] Suhr, A., Chang, M.-W., Shaw, P., & Lee, K. (2020). Exploring Unexplored Generalization Challenges for Cross-Database Semantic Parsing. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.742.

[40] Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. D. O., Kaplan, J., ... & Zaremba, W. (2021). Evaluating large language models trained on code. doi:10.48550/arXiv.2107.03374.

[41] Wei, J., Tay, Y., Bommasani, R., Raffel, C., Zoph, B., Borgeaud, S., ... & Fedus, W. (2022). Emergent abilities of large language models. doi:10.48550/arXiv.2206.07682.

[42] Popescu, A.-M., Etzioni, O., & Kautz, H. (2003). Towards a theory of natural language interfaces to databases. Proceedings of the 8th International Conference on Intelligent User Interfaces. doi:10.1145/604045.604070.

[43] Min, Q., Shi, Y., & Zhang, Y. (2019). A Pilot Study for Chinese SQL Semantic Parsing. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 3650–3656. doi:10.18653/v1/d19-1377.

[44] Tuan Nguyen, A., Dao, M. H., & Nguyen, D. Q. (2020). A Pilot Study of Text-to-SQL Semantic Parsing for Vietnamese. Findings of the Association for Computational Linguistics: EMNLP 2020. doi:10.18653/v1/2020.findings-emnlp.364.

[45] Nguyen, D. Q., & Tuan Nguyen, A. (2020). PhoBERT: Pre-trained language models for Vietnamese. Findings of the Association for Computational Linguistics: EMNLP 2020. doi:10.18653/v1/2020.findings-emnlp.92.

[46] Shi, P., Ng, P., Wang, Z., Zhu, H., Li, A. H., Wang, J., Nogueira dos Santos, C., & Xiang, B. (2021). Learning Contextual Representations for Semantic Parsing with Generation-Augmented Pre-Training. Proceedings of the AAAI Conference on Artificial Intelligence, 35(15), 13806–13814. doi:10.1609/aaai.v35i15.17627.

[47] Souza, F., Nogueira, R., Lotufo, R. (2020). BERTimbau: Pretrained BERT Models for Brazilian Portuguese. Intelligent Systems. BRACIS 2020. Lecture Notes in Computer Science, Springer, Cham, Switzerland. doi:10.1007/978-3-030-61377-8_28.

[48] Tang, Y., Tran, C., Li, X., Chen, P.-J., Goyal, N., Chaudhary, V., Gu, J., & Fan, A. (2020). Multilingual Translation with Extensible Multilingual Pretraining and Finetuning. doi:10.48550/arXiv.2008.00401.

[49] Liu, P., Yuan, W., Fu, J., Jiang, Z., Hayashi, H., & Neubig, G. (2023). Pre-train, Prompt, and Predict: A Systematic Survey of Prompting Methods in Natural Language Processing. ACM Computing Surveys, 55(9), 1–35. doi:10.1145/3560815.

[50] Turney, P. D. (2001). Thumbs up or thumbs down? Proceedings of the 40th Annual Meeting on Association for Computational Linguistics - ACL '02, 417. doi:10.3115/1073083.1073153.

[51] Vajjala, S., Majumder, B., Gupta, A., & Surana, H. (2020). Practical natural language processing: a comprehensive guide to building real-world NLP systems. O'Reilly Media, Sebastopol, United States.

[52] Lafferty, J., McCallum, A., & Pereira, F. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data. ICML, San Diego, United States.

[53] Guyon, I., Weston, J., Barnhill, S., & Vapnik, V. (2002). Gene selection for cancer classification using support vector machines. Machine learning, 46, 389-422. doi:10.1023/A:1012487302797.

[54] Och, F. J., Gildea, D., Khudanpur, S., Sarkar, A., Yamada, K., Fraser, A., ... & Radev, D. (2004). A smorgasbord of features for statistical machine translation. Proceedings of the Human Language Technology Conference of the North American Chapter of the Association for Computational Linguistics: HLT-NAACL 2004, 2-7 May, 2004, Boston, United States.

[55] Zhang, Y., & Nivre, J. (2011). Transition-based dependency parsing with rich non-local features. Proceedings of the 49th annual meeting of the association for computational linguistics: Human language technologies, 19-24 June, 2011, Portland, United States.

[56] Chollet, F. (2021). Deep learning with Python. Simon and Schuster, New York, United States.

[57] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet classification with deep convolutional neural networks. Advances in neural information processing systems, MIT Press, Cambridge, United States.

[58] Kalchbrenner, N., Grefenstette, E., & Blunsom, P. (2014). A Convolutional Neural Network for Modelling Sentences. Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics. doi:10.3115/v1/p14-1062.

[59] Kim, Y. (2014). Convolutional Neural Networks for Sentence Classification. Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP). doi:10.3115/v1/d14-1181.

[60] Radford, A., Narasimhan, K., Salimans, T., & Sutskever, I. (2018). Improving Language Understanding by Generative Pre-Training. Preprint. 1–12.

[61] Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., ... & Hon, H. W. (2019). Unified language model pre-training for natural language understanding and generation. Advances in neural information processing systems, 8-14 December, Vancouver, Canada.

[62] Yang, Z., Dai, Z., Yang, Y., Carbonell, J., Salakhutdinov, R., & Le, Q. V. (2019). XLNet: Generalized autoregressive pretraining for language understanding. Advances in Neural Information Processing Systems, MIT Press, Cambridge, United States.

[63] Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., & Zettlemoyer, L. (2020). BART: Denoising Sequence-to-Sequence Pre-training for Natural Language Generation, Translation, and Comprehension. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics. doi:10.18653/v1/2020.acl-main.703.

[64] Thoppilan, R., De Freitas, D., Hall, J., Shazeer, N., Kulshreshtha, A., Cheng, H. T., ... & Le, Q. (2022). Lamda: Language models for dialog applications. arXiv Preprint, arXiv:2201.08239.. doi:10.48550/arXiv.2201.08239.

[65] Zhang, S., Roller, S., Goyal, N., Artetxe, M., Chen, M., Chen, S., ... & Zettlemoyer, L. (2022). Opt: Open pre-trained transformer language models. arXiv Preprint. doi:10.48550/arXiv.2205.01068.

[66] Chowdhery, A., Narang, S., Devlin, J., Bosma, M., Mishra, G., Roberts, A., ... & Fiedel, N. (2023). Palm: Scaling language modeling with pathways. Journal of Machine Learning Research, 24(240), 1-113.

[67] Halevy, A., Norvig, P., & Pereira, F. (2009). The unreasonable effectiveness of data. IEEE Intelligent Systems, 24(2), 8–12. doi:10.1109/MIS.2009.36.

[68] Wang, T., Roberts, A., Hesslow, D., Le Scao, T., Chung, H. W., Beltagy, I., ... & Raffel, C. (2022). What language model architecture and pretraining objective works best for zero-shot generalization?. International Conference on Machine Learning, PMLR, 7-23 July, 2022, Baltimore, United States.

[69] Liao, Y., Jiang, X., & Liu, Q. (2020). Probabilistically Masked Language Model Capable of Autoregressive Generation in Arbitrary Word Order. Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, 263–274. doi:10.18653/v1/2020.acl-main.24.

[70] Zhao, Z., Wallace, E., Feng, S., Klein, D., & Singh, S. (2021). Calibrate before use: Improving few-shot performance of language models. International Conference on Machine Learning. PMLR, 18-24 July, 2021, Virtual Event.

[71] Holtzman, A., West, P., Shwartz, V., Choi, Y., & Zettlemoyer, L. (2021). Surface Form Competition: Why the Highest Probability Answer Isn't Always Right. Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, 7038–7051. doi:10.18653/v1/2021.emnlp-main.564.

[72] Arora, S., Narayan, A., Chen, M. F., Orr, L., Guha, N., Bhatia, K., ... & Ré, C. (2022). Ask me anything: A simple strategy for prompting language models. arXiv Preprint. doi:10.48550/arXiv.2210.02441.

[73] Lu, Y., Bartolo, M., Moore, A., Riedel, S., & Stenetorp, P. (2022). Fantastically Ordered Prompts and Where to Find Them: Overcoming Few-Shot Prompt Order Sensitivity. Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics, 1-13. doi:10.18653/v1/2022.acl-long.556.

[74] Reynolds, L., & McDonell, K. (2021). Prompt Programming for Large Language Models: Beyond the Few-Shot Paradigm. Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems, 1–7. doi:10.1145/3411763.3451760.

[75] Wei, J., Wang, X., Schuurmans, D., Bosma, M., Xia, F., Chi, E., ... & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, 28 November-9 December, 2022, New Orleans, United States.

[76] Wang, X., Wei, J., Schuurmans, D., Le, Q., Chi, E., Narang, S., ... & Zhou, D. (2022). Self-consistency improves chain of thought reasoning in language models. arXiv Preprint. doi:10.48550/arXiv.2203.11171.

[77] Kojima, T., Gu, S. S., Reid, M., Matsuo, Y., & Iwasawa, Y. (2022). Large language models are zero-shot reasoners. Advances in neural information processing systems. arXiv Preprint. doi:10.48550/arXiv.2205.11916.

[78] Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N., & Lewis, M. (2023). Measuring and Narrowing the Compositionality Gap in Language Models. Findings of the Association for Computational Linguistics: EMNLP 2023. doi:10.18653/v1/2023.findings-emnlp.378.

[79] Nye, M., Andreassen, A. J., Gur-Ari, G., Michalewski, H., Austin, J., Bieber, D., ... & Odena, A. (2021). Show your work: Scratchpads for intermediate computation with language models. arXiv Preprint. doi:10.48550/arXiv.2112.00114.

[80] Jiang, Z., Xu, F. F., Araki, J., & Neubig, G. (2020). How can we know what language models know? Transactions of the Association for Computational Linguistics, 8, 423–438. doi:10.1162/tacl_a_00324.

[81] Wallace, E., Feng, S., Kandpal, N., Gardner, M., & Singh, S. (2019). Universal Adversarial Triggers for Attacking and Analyzing NLP. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 1-15. doi:10.18653/v1/d19-1221.

[82] Shin, T., Razeghi, Y., Logan IV, R. L., Wallace, E., & Singh, S. (2020). AutoPrompt: Eliciting Knowledge from Language Models with Automatically Generated Prompts. Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), 1-15. doi:10.18653/v1/2020.emnlp-main.346.

[83] Gao, T., Fisch, A., & Chen, D. (2021). Making Pre-trained Language Models Better Few-shot Learners. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 1-15. doi:10.18653/v1/2021.acl-long.295.

[84] Davison, J., Feldman, J., & Rush, A. (2019). Commonsense Knowledge Mining from Pretrained Models. Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP), 1173-1178. doi:10.18653/v1/d19-1109.

[85] Li, X. L., & Liang, P. (2021). Prefix-Tuning: Optimizing Continuous Prompts for Generation. Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing, 1-15. doi:10.18653/v1/2021.acl-long.353.

[86] Zhong, Z., Friedman, D., & Chen, D. (2021). Factual probing is [mask]: Learning vs. learning to recall. arXiv Preprint, arXiv:2104.05240. doi:10.48550/arXiv.2104.05240.

[87] Liu, X., Zheng, Y., Du, Z., Ding, M., Qian, Y., Yang, Z., & Tang, J. (2023). GPT understands, too. AI Open. doi:10.1016/j.aiopen.2023.08.012.

[88] Gao L. (2021). On the Sizes of OpenAI API Models. EleutherAI, New York, United States. Available online: https://blog.eleuther.ai/gpt3-model-sizes/ (accessed on September 2024).

[89] Ziegler, A. (2021). GitHub Copilot research recitation. The GitHub Blog, California, United States. Available online: https://github.blog/ai-and-ml/github-copilot/github-copilot-research-recitation/ (accessed on September 2024).