Leipzig University
Institute of Computer Science
Degree Programme Computer Science, B.Sc.

Text2SQL. Exploring Relational Databases with Natural Language User Interfaces

Bachelor's Thesis

Julian Thilo
Born Oct. 27, 1995 in Münster

Matriculation Number 3797628

1. Referee: Prof. Dr. Martin Potthast

Mentor: Dr. Tim Gollub

Submission date: October 28, 2024

Declaration

Unless otherwise indicated in the text or references, this thesis is entirely the product of my own scholarly work.

Leipzig, October 28, 2024

.....
Julian Thilo

Abstract

Text2SQL is a common label for systems converting natural language input into executable SQL database queries. While SQL can be used to retrieve a broad variety of direct and indirect information from relational databases, its syntax and inner workings can be hard to learn. Hence the need for a tool to build a bridge between queries in natural language and those understood by database systems.

Coming from simple rule-based conversions, the introduction of large language models has greatly enhanced the capabilities of Text2SQL implementations. Input queries previously needed to be structured very similarly to actual SQL. The current models go beyond this approach, reducing the need to think about how a response to a query should be retrieved in advance.

This thesis evaluates the performance of two Text2SQL implementations faced with analytical questions, specifically GPT-4 and the open-source model SQLCoder-8B. The goal for the models is to provide access to meta data surrounding the information contained in the IR Anthology, a collection of papers on the subject of Information Retrieval. Real queries against this corpus and their appropriate SQL equivalents form the basis for the performance evaluation. This is in contrast to existing evaluation datasets containing less challenging questions from a wide variety of everyday topics.

The results show that the GPT-4 model can hold up to its performance on a general knowledge benchmark, while SQLCoder-8B delivers fewer correct results in comparison. During the evaluation, common errors in the generated statements are analyzed and presented as part of this thesis.

Contents

1 Introduction **1**

2 Related work **4**
2.1 Problem history 4
2.2 Benchmark datasets 5
2.3 Evaluation metrics 5
2.4 Model performance 6
2.5 Prompt engineering 7

3 Approach **8**
3.1 Dataset compilation 8
3.2 Database preparation and dataset annotation 9
3.3 Model and prompt selection 11
3.4 Generation and evaluation 13
3.5 Schema improvement and re-evaluation 14
3.6 Model temperature adjustment 15

4 Evaluation **16**
4.1 Result overview 17
4.2 Effect of temperature setting 19
4.3 Effect of schema engineering 20
4.4 Common generation errors and further observations 21

5 Discussion and outlook **24**
5.1 Dataset characteristics 24
5.2 Temperature experiments 25
5.3 Schema and prompt engineering 25
5.4 Conclusion 26

A Schema versions **27**
A.1 Initial schema for SQLCoder-8B 27
A.2 Initial schema for GPT-4 29

A.3 Extended schema for SQLCoder-8B 30
A.4 Extended schema for GPT-4 34
A.5 Modified schema for SQLCoder-8B 35
A.6 Modified schema for GPT-4 37

B Dataset questions **39**
B.1 Answerable questions 39
B.2 Unanswerable questions 41

Bibliography **44**

Chapter 1

Introduction

Information stored in databases usually goes beyond the raw data: many and potentially more significant insights can be gained by analyzing meta data, relations between different entries, and aggregate functions on top of the stored data. However, performing this kind of meta analysis requires knowledge of proper and advanced syntax of the query language SQL. Bypassing this requirement can be achieved using a means of transforming questions from natural language to proper SQL syntax automatically.

The task of transforming user input into the SQL statement required to retrieve the desired information from a database is typically labeled *Text2SQL*. Due to the prevalence of databases throughout time and different contexts, there is a long history of research on the Text2SQL task. More recently, the development of large language models (LLMs) has found its way into this research area as well.

LLMs are mostly known for the generation of natural language responses. However, these types of models may also be used to produce database queries in a structured, non-natural language, like SQL. This enables users to retrieve information from databases without knowing the proper query syntax.

Automatic query generation is especially helpful for users with limited knowledge about database queries or SQL in particular. But the implementation of Text2SQL on top of a database can also benefit more advanced users by eliminating the need to think about proper query formulation. The available information can be accessed and analyzed through straightforward, natural language user input. This increases the efficiency of information retrieval from the database.

Many general-purpose LLMs are capable of generating SQL statements from natural language input [Gao et al., 2023]. Models that have been trained specifically for this task also exist [Shi et al., 2024], trying to reduce the model size required for the task or achieve better results by cutting out general question-answering capabilities. The performance of various LLMs on the

Text2SQL task has been evaluated using the standardized benchmark dataset BIRD [Gao et al., 2023]. The benchmark shows results that go far beyond those of the approaches and models that came before LLMs [Shi et al., 2024]. The results show the potential of LLM-based Text2SQL models as natural language database interfaces. It remains unclear however whether the results carry over to a real and complex use case.

To bridge this gap, an example is needed of a database with additional information and meta data that is not readily available to users. One such database is the Webis Group's *IR Anthology*,[1] a record of publications in the field of information retrieval. The site lists conferences and journals with publications by year, allowing visitors to browse and quickly access papers they are specifically looking for. A deeper analysis of the available information and meta data requires database queries that are often very specific to an individual user's request. Anticipating all of the requests of a wide user base and manually constructing the appropriate database queries beforehand is not feasible. This is where LLM-based Text2SQL systems may be of use.

In this thesis, we evaluate one general-purpose model and one specialized model to find out whether the models can reliably produce correct SQL statements when prompted with analytical questions submitted by actual users of a real environment, the IR Anthology.

We compare the performance of the general-purpose model *GPT-4*, which scores highly on the BIRD benchmark [Li et al., 2023], and the specialized model *SQLCoder-8B*. The SQLCoder-8B model is an open-source model chosen to allow observations regarding the feasibility of low-cost implementations as compared to commercial solutions.

During the evaluation, we look for evidence of a beneficial impact of schema engineering on Text2SQL model performance, schema referring to the structuring of a database into tables and columns. To fix potential sources of common errors in the generated statements, the database structure is modified and the evaluation repeated against the new schema. Additionally, we investigate whether different settings of the temperature parameter lead to improved capabilities in the models to answer complex questions. The temperature parameter controls a model's inclination to choose tokens that are less likely as part of its generated response [Peeperkorn et al., 2024]. Hypothetically, this improves the performance on questions that require unorthodox SQL queries to be answered correctly.

Our evaluation results show that the SQL generation capabilities of GPT-4 in an analytical context are on par with the performance on the general knowledge benchmark BIRD. Unfortunately, the open-source model SQLCoder-8B

[1] `https://ir.webis.de/anthology/`

largely fails to deliver under the same conditions.

Although no statistically significant improvements are found, the evaluation provides insights into the impact of schema engineering and temperature adjustment. Additionally, this thesis highlights problems with automatic evaluation of the Text2SQL task. Further details on the model performance under different conditions are provided in the following chapters.

Chapter 2

Related work

Research into the performance of Text2SQL models has largely been focused on high-level benchmarks, evaluating models based on data from a variety of topic areas, many of which are represented by generated sample data instead of real information. This presents an opportunity for our work to evaluate model performance in a specific use case and using real data, representing a more application-oriented approach.

This chapter outlines the history of research on the Text2SQL task and describes findings and problems of existing evaluation datasets and metrics, engineering methods for the task, and the approaches used to tackle the Text2SQL problem, highlighting areas where this thesis can still contribute.

2.1 Problem history

The Text2SQL task is a longstanding topic of research in the field of natural language processing (NLP). The latest advancements from the NLP field are typically also applied to the Text2SQL problem.

By the start of the century, experiments focus on traditional NLP techniques like syntactic and semantic parsing as well as rule-based translations from parsed natural language to equivalent SQL statements (Popescu et al. [2003], Li and Jagadish [2014]). Approaches of this time split the natural language question into individual components and transform these components into SQL parts one by one.

New advances in deep learning based on neural networks were made in the field and consequently applied to Text2SQL, as summarized by Katsogiannis-Meimarakis and Koutrika [2023]: Zhong et al. [2017] adapted the *seq2seq* approach [Sutskever et al., 2014] for the problem as *Seq2SQL* and other deep learning implementations followed (Xu et al. [2017], Wang et al. [2020], Lin et al. [2020]).

The introduction of large language models (LLMs) based on transformers [Vaswani et al., 2017], the release of the BERT model [Devlin et al., 2019] and finally, the development of OpenAI's GPT models up until the release of GPT-4 [OpenAI et al., 2023] introduced a new level of interactive communicative capabilities. As Gao et al. [2024] have shown, these capabilities can also be used for improved performance on the Text2SQL problem.

2.2 Benchmark datasets

Datasets like SPIDER [Yu et al., 2018] or WikiSQL [Zhong et al., 2017] were developed before the era of LLMs to evaluate the described earlier Text2SQL approaches. These datasets remain valuable to easily evaluate the general performance of LLM-based Text2SQL models. However, they are insufficient to judge a model's capability when faced with more complex database structures and user queries.

Using a general-purpose dataset, Li et al. [2023] have shown impressive Text2SQL capabilities in the GPT-4 model and other LLMs using the newly created BIRD benchmark. The BIRD benchmark dataset is composed of over 12,000 questions asking about information from a total of 95 databases. Most of the databases contain data surrounding everyday topics like sports tournaments or company sales, mostly lacking data from less accessible fields such as academia. One exception is a single database that contains information about scientific authors, papers, conferences, and journals. Overall, the questions in the dataset can be considered more complex than those of the earlier SPIDER or WikiSQL datasets.

Due to the data stemming mostly from everyday topics, however, it remains unclear how LLM-based Text2SQL models that score highly on BIRD perform in even more complex, scientific use cases. Using a dataset built completely around scientific publication data from an actual use case, this thesis attempts to challenge the models more thoroughly.

There has been some research using domain-specific datasets, which contain information from a single field such as academic research papers (Li and Jagadish [2014], Iyer et al. [2017]). These datasets were however only used to evaluate pre-LLM models.

2.3 Evaluation metrics

For the actual evaluation task, several metrics have been established, as summarized by Katsogiannis-Meimarakis and Koutrika [2023]. The BIRD benchmark uses *execution accuracy* (EX) to decide whether the generated SQL state-

ments are correct. The EX metric compares the database result sets of both the generated and the handwritten ground truth statement. If the two sets are identical, the generated statement is labeled as correct.

The questions in our dataset are written without knowledge of the database structure, which differs from the way the BIRD dataset was compiled. The language of the questions is therefore more ambiguous than required for the reliable use of the EX metric. Between a correct LLM-generated query and the handwritten ground truth query for a question, the two result sets are not always identical due to the ambiguity of natural language. Because of this, we rely on manual examination of the generated statements to decide correctness instead of automatic evaluation using the EX metric. The problems with the EX metric and our reasons for the use of manual checks are described in more detail in Section 3.4.

2.4 Model performance

To compare the performance of different LLM-based Text2SQL approaches, the BIRD dataset is an accepted and preferred solution. Because the dataset has been purpose-built for this task, the EX measure provides workable results setting the models apart.

Researchers achieve higher EX scores on the BIRD benchmark by pairing existing LLMs with specially trained systems that provide the actual model with the information required to answer a particular question (see Section 2.5 for more details).

At the time of writing, the BIRD leaderboard is topped exclusively by approaches combining such specially trained systems with existing LLMs. The leading position is held by a proprietary solution titled *ExSL + granite-20b-code* developed by researchers at IBM, scoring at 67.86% EX.[1] The next best performing solution is *CHESS*, ranking at 66.69% EX [Talaei et al., 2024]. In third place, the *MCS-SQL* solution, built on top of GPT-4 by Lee et al. [2024], achieves a score of 65.45% EX.

The GPT-4 model is featured in a number of other solutions that have been evaluated on BIRD as well. It is also one of the few solutions that can be accessed via a programming interface, making it a preferred choice for our evaluation. When not paired with any additional system, GPT-4 achieves a score of 54.89% EX on the BIRD benchmark as evaluated by the benchmark authors themselves [Li et al., 2023].

While the GPT-4 model is available to use via a paid interface, usage of

[1] `https://research.ibm.com/blog/granite-LLM-text-to-SQL`, last accessed on 25/10/2024

an LLM for the Text2SQL task in a concrete use case may require running a model on in-house infrastructure or without payment. While a variety of open-source models that have been specifically trained for the Text2SQL task exists, these have not seen a lot of research and none are featured on the BIRD leaderboard.

However, there is one study looking at open-source LLMs for the Text2SQL problem, which saw the best performance in the SQLCoder-34B model [Zhang et al., 2024]. Because this model remains in an unfinished state as designated by the model authors,[2] we select the more recent model SQLCoder-8B from the same family for our evaluation.

2.5 Prompt engineering

To arrive at generated SQL statements that are satisfactory, the format of the prompt has to be considered and improved for the Text2SQL task.

In their Text2SQL prompt engineering study, Gao et al. [2024] show the impact of different prompt formats on a model's performance. Varying the model instructions, the embedding of the question and the format of the database structure information leads to different results from a variety of models.

Conveying the structure of the database is especially important for the generation of correct SQL statements. For large database structures, adding the whole schema to each prompt potentially exceeds the LLM's context window. LLMs can be paired with retrieval-augmented generation (RAG) layers to add only the most relevant parts of a database schema to the prompt for a specific question. As summarized by Gao et al. [2023], RAG layers can improve and compact the information available to a LLM. In the context of Text2SQL, RAG layers are trained on the database schema and documentation strings. For each question that is entered into a Text2SQL system, the RAG layer then produces the most relevant schema and documentation items and adds these to the prompt.

On the BIRD benchmark, RAG layers like MCS-SQL [Lee et al., 2024] and DAIL-SQL [Gao et al., 2024] significantly improve the scores of GPT-4 and other models. For this thesis, a freely available RAG layer is briefly considered but discarded due to incompatibility with the models used for evaluation. Our evaluation can proceed without a working RAG layer, as the schema that is used is small enough to fit into each prompt without exceeding the context window of the models used.

[2] `https://huggingface.co/defog/sqlcoder-34b-alpha`, last accessed on 25/10/2024

Chapter 3

Approach

Can Text2SQL systems be counted on to retrieve complex analytical information from a database? This is the main question we want this thesis to answer. To achieve this goal, realistic and challenging questions are sourced directly from the Information Retrieval (IR) research community. Handwritten ground truth SQL statements answering each question correctly are added to the set. Finally, the generated SQL statements for each question from two Text2SQL models are compared to the handwritten ones based on the statements and the actual database results.

3.1 Dataset compilation

Existing datasets for Text2SQL model evaluation provide a good foundation for benchmarks comparing the performance of different Text2SQL models. In the case of the most prominent candidate, BIRD [Li et al., 2023], crowd-sourced questions on a variety of different databases form a challenging dataset. This allows drawing conclusions about the general capabilities of Text2SQL models in comparison to each other.

The databases included in BIRD contain data from everyday contexts like sports tournaments or company sales. Additionally, the database structure was known to the creators of the questions. It remains unknown how the evaluated Text2SQL models perform when prompted with questions that are sourced from people without knowledge of the database structure and in a highly analytical context.

As part of this thesis therefore, a new dataset is compiled, relying on input from the IR research community. A call for questions was sent to two IR mailing lists: *SIG-IRList* by the *Special Interest Group on Information Retrieval* and the IR list curated by the *British Computer Society Information Retrieval Specialist Group*. Respondents were asked to submit questions about

IR papers and the IR research community. A total of 54 responses were sent in.

The majority of the responses contained a single question ready to be used in our dataset. The remaining responses required additional processing for one of three reasons: (1) multiple questions inside a single response, (2) questions containing placeholders instead of actual research topics or approaches, and (3) comments on the intent behind the question being sent in with the actual question.

Multiple questions were split into single dataset entries and placeholders were filled in with randomly selected IR research topics. The comments that were submitted with three of the questions did not have significant bearing on this work and were saved separately for potential future reference. The result is a list of 88 questions, including the original questions with placeholders not yet filled in.

3.2 Database preparation and dataset annotation

At the time of writing, the IR Anthology data is stored in the form of *BibTeX* files containing structured bibliographic data and not available from a relational database. For this reason, we first construct a database schema to store the information available in journal and conference bibliography files. This *initial schema* is filled with the actual data from the IR Anthology. The constructed database uses *PostgreSQL* as a database management system, because one of the two models we intend to evaluate has only been trained on PostgreSQL syntax. Preliminary testing of this model's capabilities shows that the generated SQL statements are indeed not compatible with systems other than PostgreSQL.

Many of the submitted questions are about paper citations, topics, and other information that is not contained in the bibliographic data for a paper (i.e., its BibTeX entry). To allow evaluation on these questions as well, we create a second schema, the *extended schema*. This schema includes all of the information that was asked about and could feasibly be retrieved from public sources.

Specifically, the initial schema is extended with tables for information about citations, topics, and awards as well as additional columns for information about the gender, country of origin, year of birth, and professional or academic context of paper authors and publication editors. Information about concrete research findings, the impact of certain studies or the reasoning behind the beginning and end of research into a topic is left out, because it is deemed to

Table 3.1: Dataset statistics showing the number of questions, the ratio of questions that are answerable using different schemas, and the number of questions for each complexity label

Dataset	
Total questions	88
Not answerable	41
Answerable	47
Answerable using initial schema	25
Answerable using extended schema	22
Simple questions	28
Complex questions	19

be much harder to be retrieved from public sources. This means that 41 of the 88 questions in the dataset are not answerable using the extended schema either. A statistical overview of the dataset is provided in Table 3.1.

As the actual retrieval of additional meta information is not within the scope of this work, the tables and columns that were added in the extended schema are filled with sample data. The sample data is carefully selected and crafted to allow differentiating between correct and incorrect SQL statements based on the database results.

Based on the initial and the extended database structure, each question is annotated with handwritten SQL statements that return the correct response from the respective schema. For each schema version, questions that cannot be answered using the information in that particular schema are marked accordingly.

Each question was then judged regarding its complexity using the labels *simple* and *complex*. This allows for an evaluation of the models' performance on different complexities later. A question was marked as simple if the required information is directly available in one of the database table columns and without additional calculations. In contrast, a question was marked as complex if a correct answer requires multiple sub-queries or detailed general knowledge to arrive at a calculation.

All of the database schema versions are available for reference in Appendix A.

3.3 Model and prompt selection

Our model selection is based on two goals. The first goal is to test the advertised performance of a state-of-the-art model in a real and analytical use case. The second goal is to find out whether a model that was specifically trained for the Text2SQL task and could be run on available infrastructure can hold up to the capabilities of a powerful, general-purpose model.

As mentioned in Chapter 2, the GPT-4 model [OpenAI et al., 2023] is selected to achieve the first goal. GPT-4 is at the top of the BIRD benchmark for Text2SQL systems [Li et al., 2023] at the time of model selection. Execution accuracy (EX) scores for the GPT-4 model on the BIRD benchmark range from 54.89% (zero-shot baseline) to 65.45% (paired with a specific retrieval-augmented generation layer) at the time of writing. Using the newly created dataset, the model's performance can be tested in a specific complex use case and compared to the performance on the more general BIRD benchmark.

To achieve the second goal, a version of the open-source model SQLCoder[1] is selected. The selected SQLCoder-8B model is fine-tuned for the Text2SQL task by the company Defog using Meta's Llama 3 model [Dubey et al., 2024] as a basis. It has only been evaluated on the original company's own evaluation framework, with the model authors claiming superiority over GPT-4 as well as many other open-source models in the model's description on Hugging Face[2]. A precursor to the selected model, SQLCoder-34B, has been evaluated on the BIRD benchmark by Zhang et al. [2024], achieving the best score of the tested open-source models at 32.07% EX. An instance of the selected SQLCoder-8B model is set up on university infrastructure, allowing observations regarding the feasibility of open-source Text2SQL models in an analytical context without the need for paid third-party vendors.

Both models require engineered prompts to produce desired results. For the SQLCoder-8B model, the authors have provided a prompt format (Figure 3.1) that should be used for best results, because the model is trained on it. For GPT-4, we use a prompt format from the retrieval-augmented generation (RAG) framework Vanna[3] (Figure 3.2). The Vanna framework allows Text2SQL applications to train an RAG layer on the database structure. Unfortunately, the format of the returned database objects is not well understood by GPT-4. For example, in preliminary tests, the generated SQL statements based on Vanna's schema information included the database name instead of the schema name throughout all of the responses. This renders the statements invalid, as they cannot be run against a database successfully. Instead,

[1] `https://github.com/defog-ai/sqlcoder`, last accessed on 25/10/2024
[2] `https://huggingface.co/defog/llama-3-sqlcoder-8b`, last accessed on 25/10/2024
[3] `https://github.com/vanna-ai/vanna`, last accessed on 25/10/2024

```
<|begin_of_text|>
<|start_header_id|>user<|end_header_id|>

Generate a SQL query to answer this question:
`[QUESTION]`

DDL statements:
[SCHEMA]
<|eot_id|>
<|start_header_id|>assistant<|end_header_id|>

The following SQL query best answers the question
`[QUESTION]`:
```sql
```

Figure 3.1: Prompt format for SQLCoder-8B, as recommended by the model's authors[2]

```
<system>
The user provides a question and you provide SQL.
You will only respond with SQL code and not with
any explanations.

Respond with only SQL code. Do not answer with
any explanations -- just the code.

You may use the following DDL statements as a
reference for what tables might be available:
[SCHEMA]

You may use the following documentation as a
reference as well:
The database is a Postgres database.

<user>
[QUESTION]
```

Figure 3.2: Prompt format for GPT-4, adapted from Vanna source code[3]
```

we use simple data definition language statements to provide the model with information about the database structure.

RAG layers as presented in Chapter 2 are valuable tools that are commonly used to select the most relevant parts of the database structure for each prompt. Using only a subset of the whole database structure avoids problems with context window size. The prompt is kept to a minimum length and does not contain information that is irrelevant but potentially misleading. Because the database for our evaluation is comparatively small and can be fully described using less than 1000 tokens, we opted to keep the selection or training of RAG layers outside the scope of this thesis.

The GPT-4 prompt features one more instruction defining the database system as PostgreSQL, because GPT-4 is potentially capable of generating statements in a multitude of different SQL dialects. On the other hand, the SQLCoder-8B model is only trained on the PostgreSQL dialect and does not need the additional instruction.

3.4 Generation and evaluation

Using a script, the dataset containing the questions and the handwritten ground truth SQL statement is parsed for further processing. For both the initial and the extended schema, the questions and the schema definition are then inserted into the prompts and run against both of the models. Questions that are marked as unanswerable using one of the schemas are skipped for that schema's run.

The generated SQL statements and the handwritten SQL statement for each question are run against the database. If there are any errors, these are saved for later evaluation. Otherwise, the returned database result for each generated statement is compared to the handwritten statement's result. Two indicators of similarity are calculated and added to the result summary: a flag marking responses that contain the expected result set, and the Dice coefficient [Dice, 1945] comparing the intersection size of the two result sets with the sizes of both full sets.

Using the returned results and the calculated indicators of similarity as a guiding basis, each generated statement is checked manually to decide whether it is a correct response. During this process, we also make note of common errors in the generated statements.

Manually checking the generated statements is necessary, as many of the questions in the dataset allow different interpretations of the intended results. This is inherent in the dataset as it is intentionally built on real questions from potential users of the tested system. In a theoretical user interface, the

questions would be equally ambiguous due to the characteristics of natural language [Katsogiannis-Meimarakis and Koutrika, 2023].

An exact set comparison to decide correctness (as used in the BIRD benchmark's execution accuracy measure [Li et al., 2023]) is therefore not feasible for all of the questions. A more lenient algorithm checking whether the expected results are included in a response and the Dice coefficient is sufficiently high comes to mind. Such an algorithm still lacks behind the accuracy of manual checks, as shown by the following examples from our dataset.

For the question *"What was the distribution of country of origin at last year's ECIR conference?"*, the handwritten SQL returns a list of countries, the number of participants from each country, and the percentage of that number compared to the total number of participants. In one generated SQL statement, the resulting list contains only the countries and the number of participants for each country. The percentage from the handwritten SQL result is not included in the response for the generated statement. Because the *distribution* of countries is implicit in this limited response, and the percentage was not explicitly asked for, we still mark this generation as correct.

Other times, the difference is rooted in the amount of returned results. The question *"What are the most cited IR papers from the period 2015-2020?"* refers to multiple papers without specifying the amount of papers the user is interested in. The handwritten SQL retrieves all of the papers between 2015 and 2020 sorted by their respective number of citations from the database. An LLM-generated SQL statement limiting the response to an arbitrary 10 results can still be considered to answer the question accurately and we mark it accordingly. However, not all of the expected results are included in this response and the Dice coefficient between the two is a very low 0.001477. The imagined algorithm would therefore reach the conclusion that the generated SQL is not correct.

The BIRD benchmark overcomes this problem by using over 12,000 questions for evaluation, reducing the impact of ambiguity in a subset of the questions on the overall scores. Compiling a corpus of such dimensions based on real questions regarding a specific research field is beyond the scope of this thesis. Manual result validation therefore remains the preferred if not only solution.

3.5 Schema improvement and re-evaluation

Checking the responses individually by hand allows us to identify parts of the schema appearing to lead to faulty responses. For example, the awards table in the extended schema contains both personal and paper awards, differentiated

by a reference to either a person or a paper in the respective column. This appears to lead to the models generating SQL that looks only for paper awards. We attempt to fix this and other errors by creating a new schema with modified structure.

The *modified schema* no longer contains bibliographic references that are included in the BibTeX entries but irrelevant to any of the questions in our dataset. The same is true for the table containing information about publication editors, which is removed entirely in the modified schema. To further reduce the prompt length and hopefully lead to better detection of the relevant parts of the schema, the tables for conferences and their proceedings are united into a single table. Some columns are renamed to avoid misconceptions. And the awards table is split into a table for paper awards and another table for personal awards.

Additionally, we add an instruction to both prompts informing the models that all of the database content is from the research field of IR. During the first two runs, the generated SQL statements often contain unnecessary and obstructing filters if a question mentions "IR" or "information retrieval". The added instruction (*"recall that all of the papers, journals and conferences in the database are about IR, or information retrieval"*) is an attempt to fix these faulty generations.

The generation and evaluation steps are then repeated for every question using the modified schema and prompts.

3.6 Model temperature adjustment

We expect that the models need to go beyond their typical training to find answers to the analytical questions in our dataset. Using the temperature parameter of a model, the model can be configured to choose less likely tokens for its responses more frequently [Peeperkorn et al., 2024]. We hypothesize that this may lead to better performance on complex questions. For this reason, the temperature parameter for both of the evaluated models is initially left at 0.7, which is the default for the GPT-4 model.

To test this hypothesis and as a frame of reference, the generation is repeated twice with a temperature setting of 0. This new setting means the model generations should be mostly deterministic. If the models' performance with a temperature setting of 0 remains constant and on par with the results from the previous settings, multiple runs with the previous configurations are not required to validate the evaluation results.

Chapter 4

Evaluation

Measuring the performance of large language models on the Text2SQL task in a real environment requires a different evaluation approach than in a general comparative benchmark. Benchmarks that compare Text2SQL models on a large scale typically use datasets that are less complex than in a specific analytical use case (cf. Section 2.2).

With a dataset comprised of analytical questions from actual users and a database containing scientific information on research papers, we put two Text2SQL models to a more realistic test. We evaluate how many of the generations produced by the models contain valid SQL statements and how many of those statements retrieve the correct answer to the question from the database. The results are then compared to the benchmark performance of each model.

While the overall results of the GPT-4 model are on par with the results on the BIRD benchmark, the open-source model SQLCoder-8B fails to deliver results that live up to the numbers set out by the model's authors, especially for the questions categorized as complex in our dataset.

Building on observations made during preliminary work on the dataset, we also look at the effect of structural database modifications on the quality of the generated SQL statements, running the evaluation on an *initial*, *extended*, and *modified schema* as defined in Chapter 3. Additionally, we test the hypothesis that a higher setting for the temperature parameter of the models may help with the correct generation of SQL statements to answer complex questions.

Modifying the database structure in response to observed problem areas in the models' generations leads to slight but insignificant improvements for the GPT-4 model. SQLCoder-8B shows no significant improvements in response to the modifications. Setting the temperature parameter to a higher value does not lead to improved results throughout the evaluation. A significant finding shows that the default temperature setting of 0 leads to a higher ratio

of generations containing valid SQL statements.

Finally, we identify and describe common error patterns in the SQL statements produced by the two models to provide a qualitative performance analysis.

4.1 Result overview

For each question from the dataset and each version of the database schema, we evaluate the capabilities of the two models based on two key metrics: first, whether the generated SQL statement is valid, i.e. it can be run without errors on the respective database schema. Second, if the statement is correct, i.e. it returns a correct answer to the user's question. Unlike the validity, which can be decided automatically based on the database response, the correctness is checked manually for reasons outlined in Chapter 3.

The initial schema does not contain any information beyond the data available in the IR Anthology's BibTeX files. This limits the amount of questions from the dataset that can be answered using the schema to 25. The extended and the modified schema can provide answers to 22 more of the questions, resulting in a total of 47 answerable questions. The results on the initial schema are therefore not directly comparable to the results on the extended and the modified schema.

Because of the small size of our dataset, changes in the percentages of valid and correct generations have to be interpreted with caution. The p-values for these changes, based on Fisher's exact test [Fisher, 1922], are given where applicable to indicate whether a change is statistically significant. In general, we find that the changes in the ratio of generations answering a question correctly are not statistically significant. The p-values for these changes between different schema versions and temperature settings are all higher than 0.4, meaning the null hypothesis cannot be rejected.

The results on the initial schema (as displayed in Table 4.1) are slightly better for both models with the temperature setting of 0. With this setting, the GPT-4 model generates valid SQL statements for 100% of the simple and 77.78% of the complex questions. The statements are correct for half of the simple and one of the nine complex questions. This results in a total of 92% valid and 36% correct responses.

Under the same temperature setting, SQLCoder-8B generates valid SQL statements for 84% of the questions that can be answered using the initial schema. However, only 8% of the questions are answered correctly by SQLCoder-8B. The complexity of the questions is of little consequence in this case. Only one of the simple questions and one of the complex questions are answered

Table 4.1: Model evaluation results showing the ratio of generations with valid and correct SQL for the two models on the initial schema, with highest values per model and column highlighted in bold

	Simple (n=16)		**Complex** (n=9)		**Total** (n=25)	
	valid	correct	valid	correct	valid	correct
Initial schema, temperature 0.7						
GPT-4	0.9375	0.4375	**0.7778**	0	0.88	0.28
SQLCoder-8B	**0.75**	**0.125**	0.6667	0	0.72	**0.08**
Initial schema, temperature 0						
GPT-4	**1**	**0.5**	**0.7778**	**0.1111**	**0.92**	**0.36**
SQLCoder-8B	**0.75**	0.0625	**1**	**0.1111**	**0.84**	**0.08**

correctly.

For the extended and the modified schema, the percentage of generations containing valid, and correct SQL statements varies depending on the schema and temperature used and the complexity of the question (see Table 4.2): GPT-4 generates valid SQL statements for 85.11% to 93.62% of the questions, with 38.30% answered correctly on the extended schema and 46.81% of all questions answered correctly on the modified schema. SQLCoder-8B produces valid generations for 44.68% to 72.34% of the questions. The percentage of correctly answered questions from this model ranges from 8.51% to 14.89%, both on the extended schema. On the modified schema, the model achieves correct results for 10.64% to 12.77% of the questions.

Running the evaluation on the modified schema with a temperature setting of 0, the GPT-4 model achieves its best correctness scores regardless of the question complexity. Under these conditions, the model generates correct SQL statements for 60.71% of the simple questions and for 26.32% of the complex questions. The SQLCoder-8B model achieves its best results on the extended schema with a temperature setting of 0.7 instead, with 21.43% of the simple questions being answered correctly. For the complex questions, the SQLCoder-8B results are the same for all schema variants and temperature settings. Out of the 19 complex questions, only a single question (5.26%) is answered correctly by SQLCoder-8B regardless of the schema or temperature used.

Table 4.2: Model evaluation results showing the ratio of generations with valid and correct SQL for the two models on the extended and the modified schema, with highest values per model and column highlighted in bold

	Simple (n=28)		**Complex** (n=19)		**Total** (n=47)	
	valid	correct	valid	correct	valid	correct
Extended schema, temperature 0.7						
GPT-4	0.9286	0.5	0.7368	0.2105	0.8511	0.3830
SQLCoder-8B	0.6071	**0.2143**	0.3684	**0.0526**	0.5106	**0.1489**
Extended schema, temperature 0						
GPT-4	0.9286	0.4643	**0.9474**	**0.2632**	**0.9362**	0.3830
SQLCoder-8B	**0.6429**	0.1071	**0.8421**	**0.0526**	**0.7234**	0.0851
Modified schema, temperature 0.7						
GPT-4	0.8929	0.5	**0.9474**	0.1579	0.9149	0.3617
SQLCoder-8B	0.5	0.1429	0.3684	**0.0526**	0.4468	0.1064
Modified schema, temperature 0						
GPT-4	**0.9643**	**0.6071**	0.8947	**0.2632**	**0.9362**	**0.4681**
SQLCoder-8B	0.5714	0.1786	0.7368	**0.0526**	0.6383	0.1277

4.2 Effect of temperature setting

The models are evaluated using temperature settings of 0 and 0.7 to find out if a higher temperature allows the models to come up with creative solutions required to answer some of the real questions from the dataset.

In a separate evaluation to establish the general effect of the temperature setting on the models' behavior, we find that the SQLCoder-8B model behaves deterministically with a temperature of 0, returning an identical response per question over multiple runs. On the other hand, the GPT-4 model returns identical responses for the majority but not all of the questions, with 13 of the 47 answerable questions being answered with a different response. The differences lie mostly in the naming of aliases, the ordering of table joins and columns, and the types of joins used in the statement. Of the 13 differing responses, two contain significantly different approaches to answering the question and one response has a different evaluation result compared to its counterpart from the first run.

As described already, the results on the initial schema are near-identical for both temperature settings, with slightly better performance under the 0 temperature. The improved performance is not statistically significant ($p = 0.4139$ for the improved validity and $p = 0.8031$ for the improved correctness).

On the extended and the modified schema, the difference in results is more pronounced. Running with a temperature setting of 0.7, the GPT-4 model generates valid SQL statements for 85.11% (extended) to 91.49% (modified) of all questions depending on the schema. Using a temperature setting of 0, the generated SQL statements are valid for 93.62% of the questions on both schemas. For the ratio of questions that are answered correctly, the percentage remains the same for the extended schema at 38.3% regardless of the temperature setting. For the modified schema, the percentage increases from 36.17% (temperature 0.7) to 46.81% (temperature 0). None of these improvements are statistically significant.

For the SQLCoder-8B model and regardless of the used schema, the validity scores are highest when run with a temperature of 0 ($p = 0.0049$). On the extended schema, the model generates valid SQL for 72.34% of the questions as compared to 51.06% with a temperature of 0.7 ($p = 0.0555$). On the modified schema, 63.83% of the generated SQL statements are valid, compared to 44.68% with the 0.7 setting ($p = 0.0971$). However, SQLCoder-8B's best correctness scores are instead achieved on the extended schema with a temperature setting of 0.7. 14.89% of all questions are answered correctly under these settings, compared to 8.51% on the same schema with a temperature of 0 ($p = 0.5229$), 10.64% on the modified schema with a temperature of 0.7, and 12.77% also on the modified schema with a temperature of 0 ($p = 1$).

4.3 Effect of schema engineering

For the GPT-4 model, the modifications made to the schema based on common errors in the responses lead to slightly fewer correct generations with a temperature setting of 0.7. However, with a temperature of 0, the results on the modified schema are improved compared to the extended schema, with the ratio of correct generations for simple questions rising from 46.43% to 60.71% and the ratio for complex questions remaining at the previous 26.32%.

The SQLCoder-8B model performs equally well on both schema versions. There is a slight drop in the ratio of generations containing valid SQL, with three to four fewer valid responses depending on the temperature.

None of the observed changes in correctness or validity are statistically significant with p-values ranging between 0.3754 and 1.

Positive influence of the schema modifications can be observed in the responses to questions asking about citations. For example, the column name *cited_by_paper_title*, meant to reference the title of a citing paper, is erroneously used by the GPT-4 model to look for the topic of a cited paper. After the column is renamed to *citing_publication*, it is correctly ignored and the

topic of a cited paper is sourced from the appropriate table instead.

Another change that appears to help in some cases is the reduction of complexity attempted by removing the separate table for conferences and including conference names and years in the proceedings table. For a question asking about the background of authors who contributed papers to SIGIR 2022, the GPT-4 model generates SQL looking for a conference with the name *SIGIR 2022*. On the modified schema, the title of the conference and the year are split correctly and searched for in their respective columns.

However, this particular change can also have a negative effect. For a question asking about the most influential conference about conversational search, GPT-4 generates correct SQL statements on the extended schema. On the modified schema, the statements are no longer correct, because they do not use the paper topics table anymore.

Another attempted improvement is the splitting of the awards table into two separate tables containing awards for papers and personal awards for individual authors respectively. This improves the performance of the GPT-4 model on questions asking about paper awards. For example, a question asking for a list of persons that have won the Gerard Salton Award is answered incorrectly before the change, because the singular awards table is falsely joined directly to the table containing authors and editors. After the change, the paper awards table is correctly joined via a person's authored papers.

For the same question about the Gerard Salton Award, the SQLCoder-8B already generates correct SQL statements on the extended schema. On the modified schema, the generations are incorrect, looking for a full match for the specified *Salton Award* and therefore not finding the *Gerard Salton Award* in the table.

4.4 Common generation errors and further observations

The various SQL generations by the two models show repeating error patterns. Additional observations regarding the content of the generated SQL statements can also be made.

Throughout the evaluation and regardless of the used schema or temperature, the GPT-4 model more frequently produces SQL using equality instead of partial match searches in the `WHERE` clauses. Specifically, the `=` operator is used more frequently than the `LIKE` operator combined with `%` wildcards, which is itself used more frequently than the case-insensitive `ILIKE` operator.

Searching for exact matches causes problems where the user's question contains only part of the official name of a topic or an award (as with the Gerard

Salton Award example from the previous section). It can also be problematic when the correct answer can only be retrieved using case-insensitive lookup. Because of the capitalization of paper titles, a case-sensitive filter for *argument search* does not return all relevant papers, which are much more likely to contain the phrase written with capital first letters, i.e. *Argument Search*.

This is an area where SQLCoder-8B is less likely to produce faulty results. Its generated SQL statements often make use of the `ILIKE` operator or convert both the search term and the column contents to lower case for the search.

Another common pattern is the failure to take certain keywords from the question into account for the generated SQL statements. For example, one question asks for the original paper on a topic. GPT-4 correctly generates the SQL required to retrieve papers on this topic from the database. In all evaluations except for the extended schema with a temperature setting of 0.7 however, it fails to produce the syntax to order the results by year and return only the first result (i.e., the original). Consequently, all papers on the topic are retrieved from the database instead of just the original one.

Oppositely, ignoring certain parts of the question is difficult for the models as well. Many of the generated SQL statements for the extended and the modified schema contain filters for information retrieval or IR if the question mentions IR. This behavior continues on the modified schema even though the prompt used to evaluate this schema includes a documentation string defining all items in the database as related to information retrieval or IR (as shown in Section 3.5).

The generation of invalid SQL statements frequently includes incorrect table prefixes for column names, or columns from tables that are not included in the query. Especially with a temperature setting of 0.7, the models sometimes generate syntactical features, functions, and column names that do not exist. For example, the GPT-4 model generates a query containing the non-existent `AGE` function and another containing double quotes around a string value. Double quotes in PostgreSQL signify an escaped column or table name and cannot be used to surround strings.

Other observations include the fact that different topics that were filled into an otherwise identical question can lead to entirely different responses, as can be seen in the example in Fig. 4.1. Additionally, even though the schemas used for the evaluation all use simple integer columns to store year values, both models tend to produce SQL statements using date objects to work with years. This sometimes leads to interesting though unnecessary conversions from integers to text to dates.

In which year was BERT cited the most?

```
SELECT citation_year, COUNT(*) as citation_count FROM
   paper_citations JOIN papers ON papers.id =
   paper_citations.paper_id WHERE papers.title LIKE
   '%BERT%' GROUP BY citation_year ORDER BY
   citation_count DESC LIMIT 1;
```

In which year was SVM cited the most?

```
SELECT citation_year, COUNT(*) as citation_count FROM
   paper_citations JOIN papers ON papers.id =
   paper_citations.paper_id JOIN paper_topics ON
   papers.id = paper_topics.paper_id WHERE
   paper_topics.topic = 'SVM' GROUP BY citation_year
   ORDER BY citation_count DESC LIMIT 1;
```

In which year was BM25 cited the most?

```
SELECT citation_year, COUNT(*) as citation_count FROM
   paper_citations WHERE citing_tion LIKE '%BM25%' GROUP
   BY citation_year ORDER BY citation_count DESC LIMIT 1;
```

Figure 4.1: Example showing different results for three questions that are nearly identical, as generated by the GPT-4 model for the modified schema using a temperature setting of 0

Chapter 5

Discussion and outlook

The main goal of this thesis is to find out whether Text2SQL systems can be counted on to generate SQL equivalents of complex analytical questions in a real and scientific use case. Our findings are based on the evaluation of two Text2SQL models using questions submitted by actual IR researchers. Because the questions were written without any imposed limits on their complexity, the dataset they form is more complex than generalized benchmark datasets.

We find that the open-source model SQLCoder-8B exhibits sub-par performance in our concrete setting. With a maximum ratio of 14.89% of the questions in our dataset answered correctly, using the model as part of a user interface to the IR Anthology data seems ill-advised.

The commercial model GPT-4 achieves better results, generating correct SQL statements for up to 46.81% of the questions. This is proximal to the 54.89% performance of the model on the BIRD benchmark [Li et al., 2023] used to effectively evaluate Text2SQL models in comparison to each other. Still, with less than half of the questions answered correctly, usage of the model on its own would not provide a reliable analytical user interface either.

5.1 Dataset characteristics

The evaluation dataset contains realistic questions written without detailed knowledge of the database or the parameters of the evaluation. The questions (listed in Appendix B for posterity) are therefore suited to assess the actual capabilities of Text2SQL models in a specific use case. This includes testing how the models handle the ambiguity that is inherent in natural language. However, the ambiguity of the questions gives difficulty to the automatic evaluation of the model generations.

In our case, manual evaluation of the generations was feasible due to the small size of the dataset. This comes with the trade-off of limited statistical sig-

nificance of the comparative findings of this thesis. To expand the research into the analytical capabilities of Text2SQL systems, a larger dataset is required. A larger dataset would then also require an automatic evaluation method that overcomes the problems arising with the ambiguity of natural language. The extension of our dataset and the development of a suitable evaluation method are two important opportunities for future research.

5.2 Temperature experiments

Complex questions require creative queries to retrieve the desired results. Hypothesizing that a higher setting for the models' temperature parameter might help with this, we run every evaluation once on a setting of 0.7 and a second time on a setting of 0. The results are inconclusive. SQLCoder-8B produces the highest ratio of valid SQL statements with the 0 setting and the highest ratio of correct statements with the 0.7 setting. GPT-4 is most successful with the 0 setting overall, but with little to no difference depending on the schema.

Higher temperature leads to an increased amount of SQL errors in the evaluated responses. Therefore, the temperature parameter is best kept at 0, which is the default for the SQLCoder-8B model. Building on our preliminary results, further study of the exact effects of different temperature configurations is still warranted.

5.3 Schema and prompt engineering

In our evaluation, we compare different versions of the database structure and their effect on the quality of the Text2SQL results. The evaluation results show that modifications to the schema can have a beneficial impact, as the GPT-4 model generates the highest ratio of correct SQL statements for the final improved schema. However, even though the full schema information fits inside the model's context window, invalid SQL generations still occur and the amount of incorrect generations is always higher than the amount of correct generations.

Based on current results on the BIRD benchmark, achieving a higher ratio of correct generations requires building more complex systems. The pairing of pre-trained large language models with a retrieval-augmented generation (RAG) layer has become standard praxis to score higher on the BIRD leaderboard. RAG layers allow expanding the amount of information on the structure and content of the database by selecting only the most relevant parts for each question.

Unfortunately, the freely accessible RAG layer that we selected for testing turns out to be incompatible with both of the models in our evaluation. The effect of such a layer providing only relevant information to a Text2SQL model is therefore not explored further in this thesis and open for further research. This is another area that would benefit from a larger dataset and automatic evaluation capabilities.

The same is true for experiments on the prompt format used to procure SQL statements from the models. In our evaluation, we use a single prompt format for each model continuously. Additional research can be made into the performance impact of different prompt formats in a concrete environment such as the IR Anthology. Prompt formats could be fine-tuned to the specific environment to find out how much the quality of generations improves in response.

5.4 Conclusion

In this thesis, we find analytical Text2SQL capabilities in the GPT-4 model based on a specific dataset from a real use case. In contrast, the open-source model SQLCoder-8B that is specifically trained for the Text2SQL task generates fewer correct responses. Neither of the two models is suited to be used as a drop-in solution to provide broad access to the IR Anthology's hidden meta data.

The evaluation results are nonetheless promising and provide opportunities for further research. Our work highlights the difficulties with evaluation that is based on raw human input. The problem of automatic evaluation is very much still open for future work. The dataset compiled for this thesis can serve as a starting ground for a more extensive study of the Text2SQL problem in an analytical use case.

Appendix A

Schema versions

A.1 Initial schema for SQLCoder-8B

```
CREATE TABLE public.conferences
(
id INTEGER PRIMARY KEY, -- Unique ID for each conference
year INTEGER, -- Year the conference took place in, in
   ↪ YYYY format
title VARCHAR(1000) -- Short title of the conference
);

CREATE TABLE public.journals
(
id INTEGER PRIMARY KEY, -- Unique ID for each journal
volume VARCHAR(256), -- Volume that the journal was
   ↪ published in
"number" VARCHAR(256), -- Number of the journal in the
   ↪ volume
year INTEGER, -- Year the journal was published in, in
   ↪ YYYY format
title VARCHAR(256), -- Abbreviated title of the journal
publisher VARCHAR(256) -- Publishing house of the journal
);

CREATE TABLE public.paper_authors
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that was authored
people_id INTEGER, -- ID of the person that authored the
```

```
   ↪ paper
"position" INTEGER -- Position of the author in the list
   ↪ of all authors of the paper
);

CREATE TABLE public.papers
(
id INTEGER PRIMARY KEY, -- Unique ID for each paper
title VARCHAR(1000), -- Full title of the paper
year INTEGER, -- Year the paper was published in, in YYYY
   ↪ format
bibkey VARCHAR(256), -- BibTex key of the paper
dblpkey VARCHAR(256), -- Key of the paper in DBLP library
dblpurl VARCHAR(256), -- Link to the paper in DBLP library
doikey VARCHAR(256), -- Key of the paper in DOI library
url VARCHAR(256), -- Link to the paper
venue VARCHAR(256), -- Venue the paper was published at or
   ↪  in
proceedings_id INTEGER, -- ID of proceedings the paper was
   ↪  published in
journal_id INTEGER, -- ID of journal the paper was
   ↪ published in
pages VARCHAR(256) -- Range of pages the paper was
   ↪ published in
);

CREATE TABLE public.people
(
id INTEGER PRIMARY KEY, -- Unique ID for each person
name VARCHAR(256), -- Name of the person
authorid VARCHAR(256), -- BibTex author ID of the person
editorid VARCHAR(256) -- BibTex editor ID of the person
);

CREATE TABLE public.proceedings
(
id INTEGER PRIMARY KEY, -- Unique ID for each proceedings
conference_id INTEGER, -- ID of the conference the
   ↪ proceedings belong to
title VARCHAR(1000), -- Title of the proceedings or event
year INTEGER, -- Year the proceedings were published in,
   ↪ in YYYY format
bibkey VARCHAR(256), -- BibTex key of the proceedings
dblpkey VARCHAR(256), -- Key of the proceedings in DBLP
```

```
    ↪ library
dblpurl VARCHAR(256), -- Link to the proceedings in DBLP
    ↪ library
doikey VARCHAR(256), -- Key of the proceedings in DOI
    ↪ library
url VARCHAR(256), -- Link to the proceedings
publication_series VARCHAR(1000), -- Publication series
    ↪ the proceedings belong to
volume VARCHAR(256), -- Volume of the publication series
    ↪ the proceedings were published in
publisher VARCHAR(256) -- Publishing house the proceedings
    ↪  were published by
);

CREATE TABLE public.publication_editors
(
id INTEGER PRIMARY KEY, -- Unique ID for each editor
    ↪ mapping
proceedings_id INTEGER, -- ID of the proceedings that were
    ↪  edited
people_id INTEGER, -- ID of the person that edited the
    ↪ proceedings or journal
"position" INTEGER, -- Position of the editor in the list
    ↪ of all editors of the proceedings or journal
journal_id INTEGER -- ID of the journal that was edited
);

-- paper_authors.paper_id can be joined with papers.id
-- paper_authors.people_id can be joined with people.id
-- proceedings.conference_id can be joined with
    ↪ conferences.id
-- publication_editors.journal_id can be joined with
    ↪ journals.id
-- publication_editors.people_id can be joined with people
    ↪ .id
-- publication_editors.proceedings_id can be joined with
    ↪ proceedings.id
```

A.2 Initial schema for GPT-4

```
CREATE TABLE public.conferences (id INTEGER PRIMARY KEY,
    ↪ year INTEGER, title VARCHAR(1000));
```

```
CREATE TABLE public.journals (id INTEGER PRIMARY KEY,
   ↪ volume VARCHAR(256), "number" VARCHAR(256), year
   ↪ INTEGER, title VARCHAR(256), publisher VARCHAR(256))
   ↪ ;
CREATE TABLE public.paper_authors (id INTEGER PRIMARY KEY,
   ↪  paper_id INTEGER, people_id INTEGER, "position"
   ↪ INTEGER);
CREATE TABLE public.papers (id INTEGER PRIMARY KEY, title
   ↪ VARCHAR(1000), year INTEGER, bibkey VARCHAR(256),
   ↪ dblpkey VARCHAR(256), dblpurl VARCHAR(256), doikey
   ↪ VARCHAR(256), url VARCHAR(256), venue VARCHAR(256),
   ↪ proceedings_id INTEGER, journal_id INTEGER, pages
   ↪ VARCHAR(256));
CREATE TABLE public.people (id INTEGER PRIMARY KEY, name
   ↪ VARCHAR(256), authorid VARCHAR(256), editorid
   ↪ VARCHAR(256));
CREATE TABLE public.proceedings (id INTEGER PRIMARY KEY,
   ↪ conference_id INTEGER, title VARCHAR(1000), year
   ↪ INTEGER, bibkey VARCHAR(256), dblpkey VARCHAR(256),
   ↪ dblpurl VARCHAR(256), doikey VARCHAR(256), url
   ↪ VARCHAR(256), publication_series VARCHAR(1000),
   ↪ volume VARCHAR(256), publisher VARCHAR(256));
CREATE TABLE public.publication_editors (id INTEGER
   ↪ PRIMARY KEY, proceedings_id INTEGER, people_id
   ↪ INTEGER, "position" INTEGER, journal_id INTEGER );
```

A.3 Extended schema for SQLCoder-8B

```
CREATE TABLE public.conferences
(
id INTEGER PRIMARY KEY, -- Unique ID for each conference
year INTEGER, -- Year the conference took place in, in
   ↪ YYYY format
title VARCHAR(1000) -- Short title of the conference
);

CREATE TABLE public.journals
(
id INTEGER PRIMARY KEY, -- Unique ID for each journal
volume VARCHAR(256), -- Volume that the journal was
   ↪ published in
"number" VARCHAR(256), -- Number of the journal in the
```

```
   ↪ volume
year INTEGER, -- Year the journal was published in, in
   ↪ YYYY format
title VARCHAR(256), -- Abbreviated title of the journal
publisher VARCHAR(256) -- Publishing house of the journal
);

CREATE TABLE public.proceedings
(
id INTEGER PRIMARY KEY, -- Unique ID for each proceedings
conference_id INTEGER, -- ID of the conference the
   ↪ proceedings belong to
title VARCHAR(1000), -- Title of the proceedings or event
year INTEGER, -- Year the proceedings were published in,
   ↪ in YYYY format
bibkey VARCHAR(256), -- BibTex key of the proceedings
dblpkey VARCHAR(256), -- Key of the proceedings in DBLP
   ↪ library
dblpurl VARCHAR(256), -- Link to the proceedings in DBLP
   ↪ library
doikey VARCHAR(256), -- Key of the proceedings in DOI
   ↪ library
url VARCHAR(256), -- Link to the proceedings
publication_series VARCHAR(1000), -- Publication series
   ↪ the proceedings belong to
volume VARCHAR(256), -- Volume of the publication series
   ↪ the proceedings were published in
publisher VARCHAR(256) -- Publishing house the proceedings
   ↪  were published by
);

CREATE TABLE public.papers
(
id INTEGER PRIMARY KEY, -- Unique ID for each paper
title VARCHAR(1000), -- Full title of the paper
year INTEGER, -- Year the paper was published in, in YYYY
   ↪ format
bibkey VARCHAR(256), -- BibTex key of the paper
dblpkey VARCHAR(256), -- Key of the paper in DBLP library
dblpurl VARCHAR(256), -- Link to the paper in DBLP library
doikey VARCHAR(256), -- Key of the paper in DOI library
url VARCHAR(256), -- Link to the paper
venue VARCHAR(256), -- Venue the paper was published at or
   ↪  in
```

```
proceedings_id INTEGER, -- ID of proceedings the paper was
   ↪  published in
journal_id INTEGER, -- ID of journal the paper was
   ↪ published in
pages VARCHAR(256) -- Range of pages the paper was
   ↪ published in
);

CREATE TABLE public.paper_authors
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that was authored
people_id INTEGER, -- ID of the person that authored the
   ↪ paper
"position" INTEGER -- Position of the author in the list
   ↪ of all authors of the paper
);

CREATE TABLE public.paper_citations
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that was cited
citation_year INTEGER, -- Year the paper was cited in, in
   ↪ the format YYYY
cited_by_paper_title VARCHAR(1000) -- Paper title the
   ↪ paper was cited by
);

CREATE TABLE public.paper_topics
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that contains or is
   ↪ about a certain topic
field VARCHAR(1000), -- Field that the topic is contained
   ↪ in
topic VARCHAR(1000) -- Topic that is contained in the
   ↪ paper or that the paper is about
);

CREATE TABLE public.people
(
```

```
id INTEGER PRIMARY KEY, -- Unique ID for each person
name VARCHAR(256), -- Name of the person
authorid VARCHAR(256), -- BibTex author ID of the person
editorid VARCHAR(256), -- BibTex editor ID of the person
gender VARCHAR(256), -- Gender of the person
year_of_birth INTEGER, -- Year the person was born in, in
   ↪ the format YYYY
country VARCHAR(256), -- Country the person is working in
context VARCHAR(256) -- Context the person is from, e.g.
   ↪ academia or industry
);

CREATE TABLE public.awards
(
id INTEGER PRIMARY KEY, -- Unique ID for each person
people_id INTEGER, -- ID of the person that won the award,
   ↪  or NULL if the award went to a specific paper
paper_id INTEGER, -- ID of the paper that won the award,
   ↪ or NULL if the award went to a person
award_title VARCHAR(1000), -- Name of the award
award_year INTEGER -- Year the award was won in, in the
   ↪ format YYYY
);

CREATE TABLE public.publication_editors
(
id INTEGER PRIMARY KEY, -- Unique ID for each editor
   ↪ mapping
proceedings_id INTEGER, -- ID of the proceedings that were
   ↪  edited
people_id INTEGER, -- ID of the person that edited the
   ↪ proceedings or journal
"position" INTEGER, -- Position of the editor in the list
   ↪ of all editors of the proceedings or journal
journal_id INTEGER -- ID of the journal that was edited
);

-- paper_authors.paper_id can be joined with papers.id
-- paper_authors.people_id can be joined with people.id
-- paper_citations.paper_id can be joined with papers.id
-- paper_topics.paper_id can be joined with papers.id
-- awards.paper_id can be joined with papers.id
-- awards.people_id can be joined with people.id
-- proceedings.conference_id can be joined with
```

```
   ↪ conferences.id
-- publication_editors.journal_id can be joined with
   ↪ journals.id
-- publication_editors.people_id can be joined with people
   ↪ .id
-- publication_editors.proceedings_id can be joined with
   ↪ proceedings.id
```

A.4 Extended schema for GPT-4

```
CREATE TABLE public.conferences (id INTEGER PRIMARY KEY,
   ↪ year INTEGER, title VARCHAR(1000));
CREATE TABLE public.journals (id INTEGER PRIMARY KEY,
   ↪ volume VARCHAR(256), "number" VARCHAR(256), year
   ↪ INTEGER, title VARCHAR(256), publisher VARCHAR(256))
   ↪ ;
CREATE TABLE public.proceedings (id INTEGER PRIMARY KEY,
   ↪ conference_id INTEGER, title VARCHAR(1000), year
   ↪ INTEGER, bibkey VARCHAR(256), dblpkey VARCHAR(256),
   ↪ dblpurl VARCHAR(256), doikey VARCHAR(256), url
   ↪ VARCHAR(256), publication_series VARCHAR(1000),
   ↪ volume VARCHAR(256), publisher VARCHAR(256));
CREATE TABLE public.papers (id INTEGER PRIMARY KEY, title
   ↪ VARCHAR(1000), year INTEGER, bibkey VARCHAR(256),
   ↪ dblpkey VARCHAR(256), dblpurl VARCHAR(256), doikey
   ↪ VARCHAR(256), url VARCHAR(256), venue VARCHAR(256),
   ↪ proceedings_id INTEGER, journal_id INTEGER, pages
   ↪ VARCHAR(256));
CREATE TABLE public.paper_authors (id INTEGER PRIMARY KEY,
   ↪  paper_id INTEGER, people_id INTEGER, "position"
   ↪ INTEGER);
CREATE TABLE public.paper_citations (id INTEGER PRIMARY
   ↪ KEY, paper_id INTEGER, citation_year INTEGER,
   ↪ cited_by_paper_title VARCHAR(1000));
CREATE TABLE public.paper_topics (id INTEGER PRIMARY KEY,
   ↪ paper_id INTEGER, field VARCHAR(1000), topic VARCHAR
   ↪ (1000));
CREATE TABLE public.people (id INTEGER PRIMARY KEY, name
   ↪ VARCHAR(256), authorid VARCHAR(256), editorid
   ↪ VARCHAR(256), gender VARCHAR(256), year_of_birth
   ↪ INTEGER, country VARCHAR(256), context VARCHAR(256))
   ↪ ;
```

```
CREATE TABLE public.awards (id INTEGER PRIMARY KEY,
   ↪ people_id INTEGER, paper_id INTEGER, award_title
   ↪ VARCHAR(1000), award_year INTEGER);
CREATE TABLE public.publication_editors (id INTEGER
   ↪ PRIMARY KEY, proceedings_id INTEGER, people_id
   ↪ INTEGER, "position" INTEGER, journal_id INTEGER);
```

A.5 Modified schema for SQLCoder-8B

```
CREATE TABLE public.journals
(
id INTEGER PRIMARY KEY, -- Unique ID for each journal
year INTEGER, -- Year the journal was published in, in
   ↪ YYYY format
title VARCHAR(256) -- Abbreviated title of the journal
);

CREATE TABLE public.conference_proceedings
(
id INTEGER PRIMARY KEY, -- Unique ID for each proceedings
proceedings_title VARCHAR(1000), -- Title of the
   ↪ proceedings or conference track
conference_year INTEGER, -- Year the proceedings'
   ↪ conference took place in, in the format YYYY
conference_name VARCHAR(1000) -- Short name of the
   ↪ conference the proceedings belong to
);

CREATE TABLE public.papers
(
id INTEGER PRIMARY KEY, -- Unique ID for each paper
title VARCHAR(1000), -- Full title of the paper
year INTEGER, -- Year the paper was published in, in YYYY
   ↪ format
venue VARCHAR(256), -- Venue the paper was published at or
   ↪  in
conference_proceedings_id INTEGER, -- ID of proceedings
   ↪ the paper was published in
journal_id INTEGER -- ID of journal the paper was
   ↪ published in
);
```

```
CREATE TABLE public.paper_authors
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that was authored
people_id INTEGER, -- ID of the person that authored the
   ↪ paper
position INTEGER -- Position of the author in the list of
   ↪ all authors of the paper
);

CREATE TABLE public.paper_citations
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that was cited
citation_year INTEGER, -- Year the paper was cited in, in
   ↪ the format YYYY
citing_publication VARCHAR(1000) -- Title of the citing
   ↪ publication
);

CREATE TABLE public.paper_topics
(
id INTEGER PRIMARY KEY, -- Unique ID for each mapping
   ↪ entry
paper_id INTEGER, -- ID of the paper that contains or is
   ↪ about a certain topic
topic VARCHAR(1000), -- Topic that is contained in the
   ↪ paper or that the paper is about
topic_field VARCHAR(1000) -- Field that the topic is
   ↪ contained in
);

CREATE TABLE public.people
(
id INTEGER PRIMARY KEY, -- Unique ID for each person
name VARCHAR(256), -- Name of the person
gender VARCHAR(256), -- Gender of the person
year_of_birth INTEGER, -- Year the person was born in, in
   ↪ the format YYYY
country VARCHAR(256), -- Country the person is working in
context VARCHAR(256) -- Context the person is from, e.g.
   ↪ academia or industry
```

```
);

CREATE TABLE public.paper_awards
(
id INTEGER PRIMARY KEY, -- Unique ID for each received
   ↪ paper award
paper_id INTEGER, -- ID of the paper that won the award
award_title VARCHAR(1000), -- Name of the award
award_year INTEGER -- Year the award was won in, in the
   ↪ format YYYY
);

CREATE TABLE public.people_awards
(
id INTEGER PRIMARY KEY, -- Unique ID for each received
   ↪ people award
people_id INTEGER, -- ID of the person that won the award
award_title VARCHAR(1000), -- Name of the award
award_year INTEGER -- Year the award was won in, in the
   ↪ format YYYY
);

-- paper_authors.paper_id can be joined with papers.id
-- paper_authors.people_id can be joined with people.id
-- paper_citations.paper_id can be joined with papers.id
-- paper_citations.paper_id can be joined with
   ↪ paper_authors.paper_id
-- paper_topics.paper_id can be joined with papers.id
-- paper_topics.paper_id can be joined with paper_authors.
   ↪ paper_id
-- paper_awards.paper_id can be joined with papers.id
-- paper_awards.paper_id can be joined with paper_authors.
   ↪ paper_id
-- paper_awards.people_id can be joined with people.id
```

A.6 Modified schema for GPT-4

```
CREATE TABLE journals (id INTEGER PRIMARY KEY, year
   ↪ INTEGER, title VARCHAR(256));
CREATE TABLE conference_proceedings (id INTEGER PRIMARY
   ↪ KEY, proceedings_title VARCHAR(1000),
   ↪ conference_year INTEGER, conference_name VARCHAR
```

```
    ↪ (1000));
CREATE TABLE papers (id INTEGER PRIMARY KEY, title VARCHAR
    ↪ (1000), year INTEGER, venue VARCHAR(256),
    ↪ conference_proceedings_id INTEGER, journal_id
    ↪ INTEGER);
CREATE TABLE paper_authors (id INTEGER PRIMARY KEY,
    ↪ paper_id INTEGER, people_id INTEGER, position
    ↪ INTEGER);
CREATE TABLE paper_citations (id INTEGER PRIMARY KEY,
    ↪ paper_id INTEGER, citation_year INTEGER, citing_tion
    ↪  VARCHAR(1000));
CREATE TABLE paper_topics (id INTEGER PRIMARY KEY,
    ↪ paper_id INTEGER, topic VARCHAR(1000), topic_field
    ↪ VARCHAR(1000));
CREATE TABLE people (id INTEGER PRIMARY KEY, name VARCHAR
    ↪ (256), gender VARCHAR(256), year_of_birth INTEGER,
    ↪ country VARCHAR(256), context VARCHAR(256));
CREATE TABLE paper_awards (id INTEGER PRIMARY KEY,
    ↪ paper_id INTEGER, award_title VARCHAR(1000),
    ↪ award_year INTEGER);
CREATE TABLE people_awards (id INTEGER PRIMARY KEY,
    ↪ people_id INTEGER, award_title VARCHAR(1000),
    ↪ award_year INTEGER);
```

Appendix B

Dataset questions

B.1 Answerable questions

The following list contains questions that could be answered using the information contained in at least one of the database schema versions used in this thesis (as shown in Appendix A).

- How many of the authors of SIGIR 2022 papers come from a non-academic background?
- How old was the oldest author of last year's papers?
- What was the distribution of country of origin at last year's ECIR conference?
- How many first authors are male vs. female vs. other?
- How many IR papers have a female first author?
- How many IR papers have a female last author?
- What are papers on Evaluation at ECIR 2023?
- What is the average h-index of IR people with 1000 citations in total?
- Which IR papers have received more than 250 citations?
- Who are the active community members on entity recognition in the past five years?
- Who are the active community members on topic link prediction in the past five years?

- Who are the active community members on topic web search in the past five years?
- Who did receive the most best paper awards at IR conferences?
- What is the first paper on argument search?
- What is the number of papers using "conversational search" vs. "interactive search" over the last 50 years?
- What are the most cited IR papers from the period 2015-2020?
- What is the distribution of the duration of publishing activity in the community?
- Which is the original BM25 paper?
- In which year was BERT cited the most?
- what key people should I read papers from in information filtering
- Who worked on both legal and financial search?
- Who worked on professional search and vertical search engines?
- Which people work in reproducibility?
- Who published the most papers per person on event extraction?
- What topics in the field of query understanding had more than 3 publications in 5-year periods since 2018?
- In which year was BM25 cited the most?
- In which year was SVM cited the most?
- List people that mostly focus on topic detection and tracking.
- Which persons did win the Salton Award?
- Who is the oldest author?
- How many IR papers have a female first / last author?
- Give me a list of scientists that took part in TREC more than once
- Who participated in more than one TREC track?

- Who had the most papers at SIGIR as a first author (in the past 5 years)?
- list researchers which participated in TREC and TAC as part of more than one team
- Who had the most single-authored papers at SIGIR?
- What is the distribution of number of authors for papers in the IR anthology?
- Which authors had the most SIGIR and WSDM and CIKM and WWW papers per year since 2000?
- Who is contributing the most, but cited the least?
- Who is contributing the least, but cited the most?
- Which publications received very little attention early on but became highly cited long after publication?
- Which is the most influential conference about conversational search?
- Top 3 conferences for IR?
- How many citations do best papers accumulate over 10 years, and how many did test of time award papers get?
- Which topics received a lot of early attention but were soon abandoned?
- Who was among the first researchers that worked with BERT style models?
- Which papers use monoT5 as retrieval model?

B.2 Unanswerable questions

The following questions were deemed to be unanswerable using any of the database schema versions used in this thesis (as shown in Appendix A). Some of them are included in the answerable questions in a different form, split into individual questions or with placeholders filled in.

- Gender? Age? Country? Academia or Industry?
- What are papers on Evaluation at ECIR 2024?

- What is the average h-index of IR people with 1000 / 5000 / 7500 / 10000 citations in total?
- When and where have certain concepts (or methods) emerged, how have they spread, and how have they and their meanings evolved over time or across (topical) areas?
- Which IR papers have received more than 250 / 500 / 1000 / etc. citations?
- Who are the active community members on topic X in the past five years?
- Who did receive the most best (short, reproducibility, etc.) paper awards at IR conferences?
- What are some of the major gaps in current research on IR?
- What were the most discussed problems in IR per decade?
- Which conferences are the most competitive?
- Which was the most important IR conference in each decade since the beginnings of the field?
- What was the best paper at ECIR last year?
- Most impactful trend for the next 5 years?
- What shared tasks focus on Cross-Language Information Retrieval?
- In re-ranking papers, which ranker is usually used in the first stage?
- Why are "topics" not called "queries" in ad-hoc IR?
- Who are people that curated the most-often used datasets in the IR communities?
- What were the most popular benchmarks during the period 2010-2024?
- Current IR research in three words?
- Are there papers with entries in their references lists that are not actually referenced in the paper?
- What are the trending problems in 1984 and 2024, which are still not fully solved/addressed

- Which metrics are used to evaluate conversational systems, and how frequently?
- How often is each retrieval system (Terrier, Elasticsearch, ...) used in experiments in the last 5 years?
- What are the top-50 word 3-grams used in IR papers? (with and without stopword-only phrases)
- How many (absolute + relative) references in IR anthology papers are not contained in the IR anthology?
- What are/were the top-performing open-domain question answering systems published at TREC, SIGIR or CIKM?
- What is the influence of keynotes, tutorials, presentation at summer schools, etc. on citation counts?
- Why did researchers stop pursuing approach X?
- What applications of IR help the society?
- What topics are especially interesting for PhD students looking for industry positions?
- How many GPU resources were used to produce publications at ECIR 2024?
- What sub-communities was the IR community originally composed of?
- Who have served as PC chairs at CIKM?
- How many years with IR as primary research area?
- How many years with IR as secondary research area (NLP, IA...)?
- Most relevant 'forum' for IR researchers (groups, email lists, slack, other)?
- What makes IR research unique?
- "When we talk about other thing not LLM?"
- Which researchers are active in both information science and IR?
- Most impactful innovation in the last 5 years?
- Is it possible to determine from the publication record at what point in time a problem in IR research has been solved?

Bibliography

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In Jill Burstein, Christy Doran, and Thamar Solorio, editors, *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019. doi: 10.18653/V1/N19-1423. URL `https://doi.org/10.18653/v1/n19-1423`.

Lee R. Dice. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302, 1945. doi: https://doi.org/10.2307/1932409. URL `https://esajournals.onlinelibrary.wiley.com/doi/abs/10.2307/1932409`.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, Anirudh Goyal, Anthony Hartshorn, Aobo Yang, Archi Mitra, Archie Sravankumar, Artem Korenev, Arthur Hinsvark, Arun Rao, Aston Zhang, Aurélien Rodriguez, Austen Gregerson, Ava Spataru, Baptiste Rozière, Bethany Biron, Binh Tang, Bobbie Chern, Charlotte Caucheteux, Chaya Nayak, Chloe Bi, Chris Marra, Chris McConnell, Christian Keller, Christophe Touret, Chunyang Wu, Corinne Wong, Cristian Canton Ferrer, Cyrus Nikolaidis, Damien Allonsius, Daniel Song, Danielle Pintz, Danny Livshits, David Esiobu, Dhruv Choudhary, Dhruv Mahajan, Diego Garcia-Olano, Diego Perino, Dieuwke Hupkes, Egor Lakomkin, Ehab AlBadawy, Elina Lobanova, Emily Dinan, Eric Michael Smith, Filip Radenovic, Frank Zhang, Gabriel Synnaeve, Gabrielle Lee, Georgia Lewis Anderson, Graeme Nail, Grégoire Mialon, Guan Pang, Guillem Cucurell, Hailey Nguyen, Hannah Korevaar, Hu Xu, Hugo Touvron, Iliyan Zarov, Imanol Arrieta Ibarra, Isabel M. Kloumann, Ishan Misra, Ivan Evtimov, Jade Copet, Jaewon Lee, Jan Geffert, Jana Vranes, Jason Park, Jay Mahadeokar, Jeet Shah, Jelmer

van der Linde, Jennifer Billock, Jenny Hong, Jenya Lee, Jeremy Fu, Jianfeng Chi, Jianyu Huang, Jiawen Liu, Jie Wang, Jiecao Yu, Joanna Bitton, Joe Spisak, Jongsoo Park, Joseph Rocca, Joshua Johnstun, Joshua Saxe, Junteng Jia, Kalyan Vasuden Alwala, Kartikeya Upasani, Kate Plawiak, Ke Li, Kenneth Heafield, Kevin Stone, and et al. The Llama 3 herd of models. *CoRR*, abs/2407.21783, 2024. doi: 10.48550/ARXIV.2407.21783. URL `https://doi.org/10.48550/arXiv.2407.21783`.

R. A. Fisher. On the interpretation of χ^2 from contingency tables, and the calculation of P. *Journal of the Royal Statistical Society*, 85(1):87–94, 1922. URL `https://doi.org/10.2307/2340521`.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145, 2024. URL `https://www.vldb.org/pvldb/vol17/p1132-gao.pdf`.

Yunfan Gao, Yun Xiong, Xinyu Gao, Kangxiang Jia, Jinliu Pan, Yuxi Bi, Yi Dai, Jiawei Sun, Qianyu Guo, Meng Wang, and Haofen Wang. Retrieval-augmented generation for large language models: A survey. *CoRR*, abs/2312.10997, 2023. doi: 10.48550/ARXIV.2312.10997. URL `https://doi.org/10.48550/arXiv.2312.10997`.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. Learning a neural semantic parser from user feedback. *CoRR*, abs/1704.08760, 2017. URL `http://arxiv.org/abs/1704.08760`.

George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *VLDB J.*, 32(4):905–936, 2023. doi: 10.1007/S00778-022-00776-8. URL `https://doi.org/10.1007/s00778-022-00776-8`.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *CoRR*, abs/2405.07467, 2024. doi: 10.48550/ARXIV.2405.07467. URL `https://doi.org/10.48550/arXiv.2405.07467`.

Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, 2014. doi: 10.14778/2735461.2735468. URL `http://www.vldb.org/pvldb/vol8/p73-li.pdf`.

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou,

Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-sqls. *CoRR*, abs/2305.03111, 2023. doi: 10.48550/ARXIV.2305.03111. URL `https://doi.org/10.48550/arXiv.2305.03111`.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. In Trevor Cohn, Yulan He, and Yang Liu, editors, *Findings of the Association for Computational Linguistics: EMNLP 2020, Online Event, 16-20 November 2020*, volume EMNLP 2020 of *Findings of ACL*, pages 4870–4888. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.FINDINGS-EMNLP.438. URL `https://doi.org/10.18653/v1/2020.findings-emnlp.438`.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel

Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. GPT-4 technical report. *CoRR*, abs/2303.08774, 2023. doi: 10.48550/ARXIV.2303.08774. URL `https://doi.org/10.48550/arXiv.2303.08774`.

Max Peeperkorn, Tom Kouwenhoven, Dan Brown, and Anna Jordanous. Is temperature the creativity parameter of large language models? *CoRR*, abs/2405.00492, 2024. doi: 10.48550/ARXIV.2405.00492. URL `https://doi.org/10.48550/arXiv.2405.00492`.

Ana-Maria Popescu, Oren Etzioni, and Henry A. Kautz. Towards a theory of natural language interfaces to databases. In David B. Leake, W. Lewis Johnson, and Elisabeth André, editors, *Proceedings of the 8th International*

Conference on Intelligent User Interfaces, IUI 2003, Miami, FL, USA, January 12-15, 2003, pages 149–157. ACM, 2003. doi: 10.1145/604045.604070. URL https://doi.org/10.1145/604045.604070.

Liang Shi, Zhengju Tang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *CoRR*, abs/2407.15186, 2024. doi: 10.48550/ARXIV.2407.15186. URL https://doi.org/10.48550/arXiv.2407.15186.

Ilya Sutskever, Oriol Vinyals, and Quoc V. Le. Sequence to sequence learning with neural networks. In Zoubin Ghahramani, Max Welling, Corinna Cortes, Neil D. Lawrence, and Kilian Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27: Annual Conference on Neural Information Processing Systems 2014, December 8-13 2014, Montreal, Quebec, Canada*, pages 3104–3112, 2014. URL https://proceedings.neurips.cc/paper/2014/hash/a14ac55a4f27472c5d894ec1c3c743d2-Abstract.html.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755, 2024. doi: 10.48550/ARXIV.2405.16755. URL https://doi.org/10.48550/arXiv.2405.16755.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. Attention is all you need. In Isabelle Guyon, Ulrike von Luxburg, Samy Bengio, Hanna M. Wallach, Rob Fergus, S. V. N. Vishwanathan, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 5998–6008, 2017. URL https://proceedings.neurips.cc/paper/2017/hash/3f5ee243547dee91fbd053c1c4a845aa-Abstract.html.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. RAT-SQL: relation-aware schema encoding and linking for text-to-sql parsers. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel R. Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics, ACL 2020, Online, July 5-10, 2020*, pages 7567–7578. Association for Computational Linguistics, 2020. doi: 10.18653/V1/2020.ACL-MAIN.677. URL https://doi.org/10.18653/v1/2020.acl-main.677.

Xiaojun Xu, Chang Liu, and Dawn Song. Sqlnet: Generating structured queries from natural language without reinforcement learning. *CoRR*, abs/1711.04436, 2017. URL http://arxiv.org/abs/1711.04436.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. Radev. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *CoRR*, abs/1809.08887, 2018. URL `http://arxiv.org/abs/1809.08887`.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *CoRR*, abs/2403.02951, 2024. doi: 10.48550/ARXIV.2403.02951. URL `https://doi.org/10.48550/arXiv.2403.02951`.

Victor Zhong, Caiming Xiong, and Richard Socher. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103, 2017. URL `http://arxiv.org/abs/1709.00103`.