Degree Project in the Field of Computer Science and Engineering and the Main Field of Study Machine Learning

Second cycle, 30 credits

Exploring Text-to-SQL with Large Language Models

A Comparative Study of Claude Opus and a fine-tuned smaller-sized LLM

NOA ZETTERMAN

Exploring Text-to-SQL with Large Language Models

A Comparative Study of Claude Opus and a fine-tuned smaller-sized LLM

NOA ZETTERMAN

Master’s Programme, Machine Learning, 120 credits

Date: June 17, 2024

Supervisors: Johan Boye, Jevgenij Tsoi

Examiner: Olov Engwall

School of Electrical Engineering and Computer Science

Swedish title: Generering av SQL-frågor från naturligt språk med hjälp av stora språkmodeller

Swedish subtitle: En jämförelse av Claude Opus och en finjusterad mindre språkmodell

Abstract

Using natural language to search databases is one of the oldest applications of Natural Language Processing. With the recent developments using Transformer-based Large Language Models, the possibility to convert any natural language question into SQL with high accuracy has emerged. The current State-Of-The-Art models use ChatGPT-3.5 and GPT-4 with various prompting techniques. However, the models competing with GPT-4 have not been tested. Furthermore, smaller fine-tuned models have been shown to outperform GPT-4 on specific tasks, using lower computational costs and allowing for locally hosting the model instead of using third-party APIs. This thesis compares Claude Opus, one of the competitors to GPT-4, against a fine-tuned Mixtral 8x7B on the Text-to-SQL task. Text-to-SQL entails answering a natural language question asking for data in a database, and the goal is to generate SQL that correctly answers the question. Mixtral is fine-tuned on the Big bench for large-scale databases grounded in the Text-to-SQL (BIRD) training dataset, and both models are evaluated on the BIRD evaluation dataset. The main metric used to compare the models is Execution Accuracy, which compares the resulting rows when executing queries. The two models perform insignificantly differently on the evaluation dataset, with an Execution Accuracy of 50 %, around 7 % points behind the State-Of-The-Art on BIRD. However, when analysing the data points classified as incorrect, it is found that 17 % of the evaluation dataset contain incorrect ground truth SQL queries. The evaluation also incorrectly classifies many semantically correct queries, for example, when the order of the returned rows differs, or when there are rounding differences. When accounting for the error analysis, it is found that Claude Opus significantly outperforms Mixtral, with an Execution Accuracy of 82 % and 67 %, respectively.

Keywords

Deep learning, Transformers, Natural Language Processing, Large Language Model, Fine-Tuning, Automated Database Querying, Text-to-SQL

Sammanfattning

Att använda naturligt språk för att söka i databaser är ett av de äldsta användningsområdena för naturligt-språkbehandling. Den senaste tidens utveckling med Transformer-baserade stora språkmodeller har gjort det möjligt att med hög träffsäkerhet generera SQL från frågor i naturligt språk. De modeller som är i framkant inom Text-till-SQL använder ChatGPT-3.5 och GPT-4 med olika promptningsstrategier. Däremot har inte modeller som är jämförbara med GPT-4 testats tidigare. Det har också visats att finjusterade mindre modeller har presterat bättre än GPT-4 på specifika uppgifter, samtidigt som de är mindre beräkningstunga och kan köras på lokal hårdvara istället för att bara användas via ett tredjeparts API. Denna uppsats jämför Claude Opus, en stor generell språkmodell, mot att finjustera Mixtral 8x7B på Text-till-SQL. Text-till-SQL omfattar att generera SQL baserat på en fråga ställd i naturligt språk. Mixtral är finjusterad på träningsdatamängden av "Big bench for large-scale database grounded in Text-to-SQL" (BIRD). Den huvudsakliga utvärderingsmåttet som används är exekveringsnogrannhet, som jämför om resultaten av att exekvera den genererade SQL-frågan och datamängdens referens-SQL stämmer överens. De två testade modellerna har en exekveringsnogrannhet på 50 % och är därmed inte signifikant olika när de evalueras på BIRDs evalueringsdatamängd: modellerna presterar ungefär 7 procentenheter sämre än den bästa publicerade metoden på evalueringsdatamängden. Vid en studie av de datapunkter som klassificerats som inkorrekta i evalueringsdatan visar det sig dock att 17 % av evalueringsdatan består av felaktiga referens-SQL-frågor, samt att många korrekt genererade frågor felklassificeras på grund av, bland annat, att kolumner i resultatet kommer i olika ordning, samt av avrundningsskillnader. Om man beaktar dessa fel visar det sig att Claude Opus presterar signifikant bättre än finjusterade Mixtral, med en exekveringsnogrannhet på 82 % respektive 67 %.

Nyckelord

Djupinlärning, Transformers, Naturling Språkbehandling, Stora Språkmodeller, Finjustering, Automatiserad databassökning, Text-till-SQL

Contents

List of Figures

List of Tables

List of Algorithms

List of Abbreviations

API	**A**pplication **P**rogramming **I**nterface. 3, 39, 40, 43, 45
BiLSTM	**Bi**directional **L**ong **S**hort-**T**erm **M**emory. 8, 10, 11
BIRD	**BI**g bench for la**R**ge-scale **D**atabase grounded in text-to-SQL tasks. 6, 29–31, 33–36, 39, 40, 42–47, 55, 56, 58, 61, 65
BPE	**B**yte-**P**air **E**ncoding. 14, 15, 18
CI	**C**onfidence **I**nterval. 47, 48, 51
CLT	**C**entral **L**imit **T**heorem. 48
CoT	**C**hain **o**f **T**hought. 13, 34
DBMS	**D**ata**b**ase **M**anagement **S**ystem. 5
DDL	**D**ata **D**efinition **L**anguage. 9, 35, 42, 45, 52
EM	**E**xact **M**atching Accuracy. 27, 30, 37
EX	**EX**ecution Accuracy. 27, 28, 30, 35–37, 39, 47, 51–54, 58
FS	**F**ew-**S**hot. 13, 25, 33–37, 43, 66
FT	**F**ine-**T**uned. 22–24, 35–37, 39–43, 47, 51–55, 57, 60, 62, 79, 84
GNN	**G**raph **N**eural **N**etwork. 10
GPT	**G**enerative **P**re-trained **T**ransformer. 13, 15

GPU	**G**raphics **P**rocessing **U**nit. 2, 22, 24, 39, 46, 62
IID	**I**ndependent and **I**dentically **D**istributed. 48
LLM	**L**arge **L**anguage **M**odel. 1–3, 5, 7, 8, 10–14, 16, 18, 22, 25, 26, 29, 32, 33, 36, 39, 40, 44, 45, 55, 56, 62
LoRA	**Lo**w-**R**ank **A**daptation. 23, 24, 45
MoE	**M**ixture **o**f **E**xperts. 25
NLP	**N**atural **L**anguage **P**rocessing. 1, 7, 16
PEFT	**P**arameter **E**fficient **F**ine-**T**uning. 10, 22
PUE	**P**ower **U**sage **E**ffectiveness. 62
RAG	**R**etrieval **A**ugmented **P**rompting. 34, 35
RAM	**R**andom **A**ccess **M**emory. 24
RNN	**R**ecurrent **N**eural **N**etwork. 16
RoPE	**Ro**tary **P**ositional **E**mbeddings. 16
SMoE	**S**parse **M**ixture **o**f **E**xperts. 23, 25, 26, 41, 42, 46
SOTA	**S**tate-**O**f-**T**he-**A**rt. 2, 7, 13, 52
SQL	**S**tructured **Q**uery **L**anguage. 1, 2, 5–7, 9–13, 24–31, 33–36, 39, 40, 42–45, 47, 52, 55, 58–61, 65, 81, 90
VES	**V**alid **E**fficiency **S**core. 28–30, 36, 39, 47, 51, 52
VRAM	**V**ideo **R**andom **A**ccess **M**emory. 2, 22, 24, 26, 39, 46

Chapter 1

Introduction

Using natural language to search databases is one of the oldest applications of Natural Language Processing (NLP) [1]. The introduction of Large Language Models (LLMs) have drastically improved the ability to transform natural language queries into executable Structured Query Language (SQL). The task of converting natural language into SQL is known as Text-to-SQL. Introduction to what text-to-sql is more in depth with an example

The relational database model was originally developed to make it easier to interact with databases by removing the need to know how data is stored in hardware [2]. SQL is a programming language built on the relational database model and it allows anyone familiar with the language to write queries to a database without knowledge of the underlying structure of the stored data. Text-to-SQL aims to add a another layer of abstraction, where the language used to request data is natural language instead of SQL, allowing anyone to retrieve data from a database. The natural language question is converted into SQL by a language model that knows the structure of the database. Finally, the generated SQL can be executed and a result can be retrieved. The goal of the language model is only to convert the natural language question to an SQL query which when queried against a database answer the natural language question.

Good Text-to-SQL capabilities have profound implications for data analysis and the democratization of data access, allowing users with no programming knowledge to extract insights from complex databases. Despite recent advancements using LLMs for Text-to-SQL, the challenge of accurately interpreting and converting natural language to SQL is far from solved. The current State-Of-The-Art (SOTA) models are still far from human expert performance [3, 4, 5] and cannot reliably be used without carefully reviewing the generated SQL. A short introduction to how SQL works is provided in Section 2.1.

The current SOTA model used for Text-to-SQL is GPT-4 [6], with various techniques to ensure coherent SQL generation [4, 5, 7]. However, GPT-4 is proprietary, uses a lot of resources, can be expensive to use and has limitations with respect to fine-tuning to further increase performance for Text-to-SQL. In contrast, there exist open-sourced models which are smaller, more resource efficient and can be fine-tuned to specialize its knowledge at a specific task but with less generalization capabilties than GPT-4. The smaller, fine-tuned models have been shown to outperform GPT-4 at specific tasks [8, 9]. This raises the question if fine-tuned open-source models can outperform the best proprietary LLMs at the Text-to-SQL task.

1.1 Research Question

This thesis aims to compare fine-tuning of a *smaller* LLM and the proprietary language model Claude Opus. A *smaller* LLM is considered as one which can be fine-tuned on a Graphics Processing Unit (GPU) limited to using 80GB of Video Random Access Memory (VRAM). And the selection of proprietary language model is motivated in Section 4.2.1

Research Question: Is there a difference in performance when comparing a *smaller* fine-tuned LLM and Claude Opus on the Text-to-SQL task?

1.1.1 Hypothesis

Null Hypothesis H_0: There is no significant differences between the models. Alternative Hypothesis H_a: There is a significant difference between the two models.

1.2 Limitations

The main limitation is with respect to compute, fine-tuning LLMs is a compute intensive task, which require expensive GPU hardware. The company have offered to provide computational cloud resources for fine-tuning on an A100 GPU with 80GB VRAM.

The study is limited to only using one model for fine-tuning and only one proprietary LLM on only one dataset. Furthermore, only having Application Programming Interface (API) access for the proprietary LLM can hinder flexibility, for example with less options to customize the selection of tokens in output generation.

Chapter 2

Background

This chapter presents the relevant background to understand the thesis, covering SQL, what Text-to-SQL is and various techniques used for Text-to-SQL. Furthermore, LLMs and the Transformer architecture is covered to help understand how the current models used in Text-to-SQL work. Some techniques to increase inference speed and decrease training requirements for LLM, used in the thesis, are then covered. Finally, the metrics used for Text-to-SQL and relevant datasets are presented.

2.1 Databases & SQL

To store data on disk in a structured way, databases are used. These databases are managed by Database Management Systems (DBMSs), which has been used since the 1960s [10]. There are a few such systems popular to this day, one of them being the Structured Query Language (SQL) which is a programming language used to retrieve data from a database. SQL uses the relational database model introduced by [2]. The aim of the relational database model is to add an abstraction layer that allows the database user to use the database without knowledge of how the data is stored on disk, ultimately making databases easier to use.

The relational database model expects each row in a table to be unique. This is enforced by using a primary key in each table, unique for each row in the table. The key can be created from existing columns, or be a new column with a generated unique identifier. To reference a row in a table, a foreign key is used. The foreign key is a copy of the primary key in the referenced row and table, such that the two can be linked together. In SQL, tables can be joined together based on specific fields by using the JOIN keyword. Keywords are all words which are reserved for usage in the SQL language. Perhaps the most widely used keywords are SELECT and FROM keywords. An example will be used to explain how parts of the SQLite language works. The example uses the SQLite dialect and the formula_1 database in the BIRD dataset, that is also used in most Text-to-SQL datasets.

```sql
SELECT c.constructorRef, c.url
FROM results r
JOIN constructors c
ON r.constructorId = c.constructorId
WHERE r.raceId = (
  SELECT raceId
  FROM races
  WHERE year = 2009
  AND name = 'Singapore Grand Prix'
) AND r.position = 1;
```

The example uses the Formula 1 database, and each part of the query as well as the necessary understanding of the tables and columns will be explained.

`results`, `constructors` and `races` are tables. `results r` means that columns in the results table can be accessed via the alias `r`, such as `r.position`. Using `results AS r` is also valid syntax to create an alias. The `constructorId` is the primary key in the `constructors` table, and joining the `results` and `constructors` table on `constructorId` means that the two tables are merged into one, such that each row in results with ID X will also contain the `constructors` row with ID X. For example, if three rows in the `results` table contain `constructorId = 5`, then all those three rows would be joined with three copies of the row in the `constructor` table with `constructorId = 5`.

The `(SELECT ...)` in `WHERE r.raceId = (SELECT ...)` is a separate query from the main query, it selects the `raceId` from the `races` table, with conditions applied to the `races.year` and `races.name` columns. The resulting table produced by such inner query can in general return multiple rows and values. However, in this example it is assumed that the Singapore Grand Prix race was only held once in 2009. The reason why this is important is because the equality sign in `r.raceId = (SELECT...)` only matches on row from the inner `SELECT` statement. If all returned rows should be considered, the `IN` keyword can be used instead of `=`.

Putting the entire query together, the website url and constructor reference name of the racer who came in first position of the 2009 Singapore Grand Prix is selected. Although only information from the constructors table is selected from the `SELECT` statement, both the results and races tables must be used to filter out the right constructor based on the conditions.

Some other common SQL keywords are `ORDER BY`, which sorts the rows in ascending order based on a column value. The `DESC` keyword can be used to specify that the ordering should be done descending instead of ascending. Furthermore, there exist multiple ways to join tables. In SQLite, `JOIN` and `INNER JOIN` both function the same, which is explained in the example. The other join statements are `LEFT JOIN` and `RIGHT JOIN`, which keeps rows which are not linked from one of the tables. Further reading about how SQLite works can be done on sqlite.org.

2.2 Natural Language and Database querying

Using natural language to search databases is one of the oldest applications of NLP. Methods such as using a restricted subset of natural language with explicit rules to convert the text into database queries [1], and specifying relationships between data and categories [11] were some strategies used to search databases with natural language in the 70s and 80s. These old methods were often specific-purpose and hand-crafted to work for a specific database.

In the early 2000s, methods had evolved to handle a wider range of natural language and the system could understand when the questions were too vague for it to answer [12]. The current SOTA models for interacting with databases using natural language use the transformer architecture. LLMs, together with various techniques to represent the database to the model and generate the SQL has improved the performance of Text-to-SQL a lot in recent years.

The Text-to-SQL task consists of predicting an SQL query from a natural language question. The problem can be divided into two parts, knowledge representation and output generation. Knowledge representation entails the database structure and how that structure is provided to the language model, and output generation is the step which generates SQL. The ground truth query is called the *gold query*, and the natural language question will be referred to as *utterance*, although it is written text.

2.2.1 Knowledge Representation

The language model must have knowledge of the database structure in order to make predictions about the database. In older systems, the database representation had to be designed by hand [1, 11]. But now, there are general methods to create a knowledge representation regardless of the database structure. These methods will be presented in this section.

2.2.1.1 Schema Filtering

Schema Filtering is the process of removing unnecessary tables and columns, given an utterance and a database. In almost all utterances, at most a few tables are used, and only a few columns within each table. The schema filtering is done in order to shorten the length of the input to the model in the main prompt. LLMs has been shown to perform worse at longer contexts [13], with parts of the input sequence being ignored when generating the output. Databases can often have many tables, with multiple columns in each, making the schema long when written out in text. Filtering is therefore used in order to shorten the context of the schema, while keeping the relevant information. A general method to use schema filtering with LLMs is to prompt the LLM in order to filter relevant tables and columns.

2.2.1.2 Schema as Text

One way to represent the database schema to the language model is by creating a representation of the database schema in plain text and use this text together with the utterance as input to the language model.

SQLNet [14] was the first Bidirectional Long Short-Term Memory (BiLSTM) model to do this. It used the WikiSQL [15] dataset, which only contains one table per database. The names of the columns are passed to the BiLSTM together with the question.

SyntaxSQLNet [16] also uses an BiLSTM. However, unlike SQLNet the knowledge representation is designed to handle multiple tables. This is done by providing the table name, column name, and type information about the column, unlike SQLNet, which only provides the column names.

In transformer-based models, there are two popular text representations of the database, using SQL `CREATE TABLE` statements (Data Definition Language (DDL) Format) or using a Compact Notation. The two formats are shown in Table 2.1. The DDL Format can give more information to the model, while the Compact Notation does not use as many tokens. Most models that use the Compact Notation add a separate section in the input text to specify primary and foreign key relations. With both DDL Format and Compact Notation, examples of column values may also be provided to give the model further information about the database and what the column values may look like.

Table 2.1: The DDL Format and Compact Notation format compared on the same example table. SQLite DDL Format is used as DDL format.

DDL Format	**Compact Notation**
`CREATE TABLE Employee (`	`Employee(`
`  EmployeeID INTEGER NOT NULL,`	`  EmployeeID,`
`  FirstName TEXT,`	`  FirstName,`
`  LastName TEXT,`	`  LastName,`
`  Position TEXT,`	`  Position,`
`  GroupID INTEGER,`	`  GroupID`
`  PRIMARY KEY (EmployeeID),`	`)`
`  FOREIGN KEY (GroupID)`	
`    REFERENCES Group(GroupID)`	
`);`	
Explicit schema definition with data types, constraints and keys.	Compact summary of the schema with only table and column names.

2.2.1.3 Relational Graphs

Relational Graphs is a knowledge representation method where the database is represented as a graph, where columns and tables are represented as nodes and directed edges correspond various types of logical links between the nodes [17]. For example, columns that belong to a table can have a `BELONGS-TO` link. If a column in table B is a foreign key of a column in table A, the columns may have a `FOREIGN-KEY` relation. The edge names differ between implementations, but the key idea is that edges are directed and have labels that specify the connection between nodes. The utterance can also be converted into the relational graph. The words in the utterance that match with names of columns or tables are given edges stating a match relation, and the words within the utterance can be given edges based on dependency grammar [18], for example specifying adjective modifiers, objects of a verb and more.

Perhaps the most natural way to incorporate the relational graph into a neural network is by using a Graph Neural Network (GNN) such as in [19]. Most methods using relational graphs use a relational-aware transformer, also considered a GNN. For the transformer to correctly use the graph as input, relational-aware self-attention is used [20]. An in depth introduction of normal attention is covered in Section 2.7. This method is also used in various Text-to-SQL methods [17, 21, 22, 23]. To enforce the relational structure in the attention mechanism, a relational matrix, r, is added to the Key and Value matrices in Equation 2.4. Each position, $r_{i,j}$ in r is a vector embedding that is selected depending on the type of directed edge from node i to node j in the relational graph.

The graph-aware attention model can be used as the main model [17, 21], but also be combined with an existing pre-trained transformer [22, 23]. In the first case, the entire model must be trained from scratch, and in the other only the additional weights added by the relational-aware self-attention must be trained from scratch, at the same time as fine-tuning the pre-trained model. The RASAT [22] and Graphix-T5 [23] models utilize pre-trained transformers and add additional relational-aware self-attention weights, which in RASAT is around $0.01\,\%$ of the total model parameters. However, unlike Parameter Efficient Fine-Tuning (PEFT) methods, both the additional weights and the pre-trained transformer weights must be trained together to get good results.

2.2.2 Output generation techniques

There exist a variety of techniques for generating SQL queries with language models. The methods using BiLSTMs use techniques to allow the language model to focus on the logic rather than the grammar of the language when generating the query. The methods using transformer models and LLMs have adopted some of the techniques used in BiLSTMs, as well as implemented techniques shown to increase the performance of transformers, such as Beam Search. Some completely new generation techniques with LLMs for the Text-to-SQL task have also been developed, for example PICARD [24].

2.2.2.1 SQL Sketch

The SQL Sketch was first introduced by SQLNet [14]. It allows for the model to only predict relevant keywords in the SQL query, instead of both predicting the grammar and keywords when generating SQL. This is done by first writing a general layout of the SQL query where table names and logic is left to be predicted by the language model. For example, a simple `SELECT` and `FROM` sketch that supports selecting columns from a table would look like:

```
SELECT ($COLUMN)+ FROM $TABLE
```

Where `COLUMN` and `TABLE` are keywords to be predicted by the language model. The + after `($COLUMN)` represents that there may be one or more column keywords used, and a * indicates that zero or more of the clause in brackets may be used. The sketch designed in [14] is:

```
SELECT $AGG $COLUMN WHERE $COLUMN $OP $VALUE
(AND $COLUMN $OP $VALUE)*
```

This sketch is designed to work for all queries in the WikiSQL [15] dataset, where each database only contain one table. However, [14] states that the sketch approach is generalizable to include all possible SQL queries.

2.2.2.2 SQL Grammar and Syntax Trees

One way to restrict generated output to valid SQL is to use grammars and abstract syntax trees. Grammar and Syntax Tree based methods design a grammar for the SQL language. The methods are used in both BiLSTM [16, 17, 25] models and LLMs [26].

However, restricting generation to the grammar is insufficient for generating valid SQL, as there are some logical constraints on top of the grammar-decoding process [25]. Furthermore, to generate valid SQL with respect to the current database, the table and column names must be incorporated into the grammar, which can be done automatically. Most Text-to-SQL methods that use a grammar-based generation also apply these methods to ensure that valid SQL is generated. The SQL grammar methods can also be used together with other techniques such as Beam Search [24, 26].

2.2.2.3 Beam Search

Beam Search is a method that is widely adopted in LLMs [27, 28, 29, 30] and has also been used in LLMs for the Text-to-SQL task [23, 24, 26, 31].

Beam Search is an algorithm that finds a good solution in a search tree. It utilizes a breadth-first search with a limited number of branches to consider at any time. To determine which states to keep, a heuristic is used to determine which states are most promising. In LLMs, the children of a node in the tree consist of all different combinations of output tokens, and the heuristic score used is usually the probabilities of the tokens that are generated by the model, or some combination of the probabilities over multiple generated tokens.

For the Text-to-SQL task, there may be further checks such as in PICARD [24], which removes some of the k selected tokens where the generated SQL would be invalid.

In [32], it is found that that Beam Search works poorly when generating code due to the generation getting stuck in loops, generating the same code over and over. However, models such as Codex that are accessed through an API do not support Beam Search [33]. On the contrary, PICARD [24] has successfully implemented Beam Search in the open-sourced T5 model and shows that Beam Search works well with a strict restriction on the output tokens to prevent incorrect SQL from being generated.

2.2.2.4 Self-Correction

A common prompting technique used with LLMs for programming languages is to re-prompt the language model with incorrectly generated output and the produced error until the generated output is syntactically correct, allowing the model to fix some of its own mistakes. There exist a few different methods using self-correction in Text-to-SQL [4, 7, 34]. In general, the technique starts with an incorrect SQL query and prompts the LLM with the query and the error generated when executing the query. Additionally, information regarding the database schema and question can also be included. Then, a new SQL query is generated based on the prompt. If it is incorrect again, the same procedure is repeated for a limited number of tries.

2.2.2.5 Self-Consistency

LLMs are often used in a non-deterministic way, where the next token is selected based on the probability distribution of the generated output.

Therefore, when generating an answer to the same question multiple times, the LLM may generate different responses. The self-consistency method builds on stochastic sampling, generating multiple sequences from the same prompt by sampling from the probability distribution at each generation step, obtaining multiple answers to the question. And in a final step, selecting the most common answer among the sampled answers. Furthermore, the originally proposed self-consistency method [35] uses stochastic sampling together with Few-Shot (FS) Chain of Thought (CoT) [36] prompting.

CoT is a method that encourages the model to reason before outputting the final answer. This is usually done by giving FS examples where the answers contain reasoning steps, encouraging the model to generate reasoning steps as well. This method is shown to improve LLMs performances on tasks which may require reasoning before giving a final answer to the question [36], for example in mathematics and programming.

Multiple successful Text-to-SQL methods have drawn inspiration from the self-consistency method [37, 38], with the main difference being that the execution result from executing the generated SQL outputs are compared, instead of the queries themselves. This is done since two different SQL queries may yield the same result and carry the same semantic meaning.

2.3 Large Language Models

As shown in Section 2.2, recent methods in Text-to-SQL use transformer-based LLMs. LLMs are language models which have a lot of parameters, typically at least billions of parameters. The most common architecture for these models are variants of the transformer architecture. LLMs have been shown to perform and generalize well, and current models [6, 39] are on par with human experts on a wide variety of professional and academic exams.

The Generative Pre-trained Transformer (GPT) series of models [6, 40, 41, 42] from OpenAI use a decoder only transformer. The decoder only transformer is autoregressive and trained using next-word prediction. However, this does not limit the model functionality compared to encoder-decoder models. The user input and already generated output is used as input to the decoder. GPT1 [41] was the first successful language model to generalize across tasks and at the time outperform SOTA specific-purpose models.

To understand how LLMs work, the Transformer architecture will be explained. Starting by explaining word embeddings and positional encodings, which is used as input to the model, and following with the attention mechanism which is the main building block of the transformer. Finally, the entire transformer architecture is put together.

2.4 Tokenization

Tokenization is together with word embeddings the two steps required to transform text to vectors of numbers that a LLM can use. The tokenization process splits a text into tokens. Tokens can be words, characters or subwords, depending on the selected tokenization method. Some of the simpler methods include whitespace tokenization and character tokenization, where whitespace tokenization splits the text into tokens based on whitespace and character tokenization uses characters as tokens. However, there are also some more sophisticated methods such as Byte Pair Encoding (BPE) and WordPiece tokenization. These methods are used for tokenization in LLMs and the tokens are subwords, which are learned from a training corpus.

The objective that is optimized when selecting which tokens to use in WordPiece [43] is to tokenize the training corpus into as few tokens as possible, using at most K word or subword tokens [44]. The algorithm starts by splitting the corpus into words based on whitespace and punctuation and using all existing characters in the corpus as tokens. After that, to create the final K word pieces, existing tokens are combined to create new tokens, where in each iteration the token pair with the highest score according to Equation 2.1 is selected as a new token.

$$\text{score} = \frac{\texttt{freq\_of\_pair}}{\texttt{freq\_of\_first\_element} \times \texttt{freq\_of\_second\_element}} \tag{2.1}$$

To keep track of which tokens start a word and which are not the start of a word, special characters are used. For example, the tokenization of the word bird may be split as ($\texttt{\_bi}$, $\texttt{\#\#rd}$), where in this case _ is used to indicate the start of a word and ## indicates that the token is the continuation of a word. To tokenize a text with the WordPiece tokenization, each word is split recursively on the longest subword token that exist within that word. If a word cannot be fully tokenized, the entire word is tokenized by an `UNK` (unknown) token. The only case when a word cannot be fully tokenized is when the word contains a character not seen in the training corpus. One of the common pre-trained word embeddings used in transformers, BERT [45] uses the WordPiece tokenization technique.

The other popular technique in transformer models are BPE, originally introduced as a data compression algorithm [46, 47]. It has since been adapted to machine translation tasks. The algorithm to create the tokens works by creating new tokens based on the most common pairs of existing tokens until a threshold for the max number of tokens is reached. The difference from WordPiece is that BPE only uses `freq_of_pair` as the score, without dividing by the frequencies of the individual elements. The tokenization process of a text with BPE is similar to WordPiece, except that the `UNK` token doesn't replace the entire word, but instead only the unknown subword. BPE is used in models such as the GPT models [40, 41, 42] and BART [48].

2.5 Word Embeddings

Word embeddings represent tokens in vector space in order to help learning algorithms understand natural language [49]. The method for learning the embeddings can be split into two categories, those which are pre-trained using an algorithm designed to learn word vectors, such as Word2Vec [49] and GloVe [50] and embeddings which are learned while training the model that is using the embeddings as input. The learned embeddings are used in the transformer architecture presented in Section 2.8, where the embeddings are learned via backpropagation, and get updated like any other weights in a neural network.

2.6 Positional Encodings

Positional encodings are used to encode the position of a word embedding in a text. This is mainly done in order to add positional information to the input to a model which does not explicitly consider the order of the input, such as the transformer. The positional encoding allows the model to understand the order of the tokens in the text even though the tokens are represented as a set.

In [51], a sinusoidal positional encoding is used:

$$\text{PositionalEncoding}_{(pos,2i)} = \sin\left(\frac{pos}{10000^{2i/d_{\text{embedding}}}}\right) \tag{2.2}$$

$$\text{PositionalEncoding}_{(pos,2i+1)} = \cos\left(\frac{pos}{10000^{2i/d_{\text{embedding}}}}\right) \tag{2.3}$$

where i corresponds to the element in the positional embedding vector, which has the same shape as the word embedding. pos is the position of the token. The sinus function was chosen to allow the model to learn to attend to relative positions of tokens as $\text{PositionalEncoding}_{pos+k,i}$ can be represented as a linear function of $\text{PositionalEncoding}_{pos,i}$ [51]. The positional encodings are added to the word embeddings before being used by the model.

Alternatively, the positional information can be learned during training. The learned vectors are called positional embeddings instead of encodings. Positional embeddings shows similar results to the sinusoidal encodings [51]. However, it has also been shown that no positional encoding or embedding layer is required in auto-regressive transformers [52].

There also exist newer techniques which use positional embeddings: [53, 54, 55], which show slight improvements over the original sinusoidal encoding. For example, Rotary Positional Embeddings (RoPE) [54] attempts to encode information such that only the relative position between any two tokens is kept when attention is applied. And [53] tests a Gaussian embedding, utilizing the normal distribution instead of the sinusoidal function.

2.7 Attention

The main building block of the Transformer architecture, which is used in almost all LLMs is attention. However, attention was initially developed to allow for a Recurrent Neural Network (RNN) to automatically search the source sentence relevant for predicting the next word, a mechanism known as attention is introduced [56]. In the context of NLP, attention allows for the model to learn to focus on relevant tokens in the input sentence when predicting the next token. The attention introduced by [56] used in RNNs is an additive attention. However, the attention used in LLMs is a dot-product attention. Scaled dot-product attention [51] is the main building block of the transformer architecture. The scaled dot-product attention is defined by:

$$\text{Attention}(Q, K, V, M) = \text{Softmax}\left(\frac{QK^T + M}{\sqrt{d_{key}}}\right)V \tag{2.4}$$

$$\text{Softmax}(x_{ij}) = \frac{e^{x_{ij}}}{\sum_k e^{x_{ik}}} \tag{2.5}$$

where $Q \in \mathbb{R}^{T\times d_{key}}$, $K \in \mathbb{R}^{T\times d_{key}}$ and $V \in \mathbb{R}^{T\times d_{value}}$ are called Query, Key and Value, T is the number of tokens. $M \in \mathbb{R}^{T\times T}$ is an optional mask which can be used to ignore certain Key-Query pairs after the Softmax operation.

In the case of natural language, the rows in the Query, Key and Value are all vector representations related to each token. The idea is that the result from attention is added onto the vector representation to add contextual information regarding each token. Before giving an interpretation of attention, Multi-Head Attention will also be introduced, which also introduce trainable weight matrices, which is used to learn a transformation from the vector embedding representations to the d_{key} and d_{value} dimensional representations.

Multi-Head Attention [51] is when a linear transformation followed by attention is applied multiple times in parallel, each attention operation is called an attention head. And finally combined through another linear transformation which downscale the computed concatenated attention heads. Multi-Head attention is defined as:

$$\text{MultiHead}(Q, K, V, M, h) = \text{Concat}(\text{head}_1, \text{head}_2, \ldots, \text{head}_h)W^O \tag{2.6}$$

$$\text{where head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V, M) \tag{2.7}$$

where h represent the number of attention heads, and the attention heads are concatenated into dimensionality $\mathcal{R}^{T\times hd_{value}}$. And $W_i^Q \in \mathcal{R}^{d_{\text{model}}\times d_{key}}$, $W_i^K \in \mathcal{R}^{d_{\text{model}}\times d_{key}}$, $W_i^V \in \mathcal{R}^{d_{\text{model}}\times d_{value}}$, $W^O \in \mathcal{R}^{hd_{value}\times d_{\text{model}}}$ are learnable weight matrices and T is the number of tokens.

Analysis of the attention heads have shown that different attention heads learn to accurately attend to different aspects of the language, such as attending to periods or direct objects attending to their respective verbs and other grammatical structures [57].

Focusing on how one attention head works, the Key, Query and Value used in Equation 2.4 are transformations of the model vector embeddings for each token, generally to a smaller size compared to the model embeddings, which can be thought of as focusing on a specific aspect of the tokens in that attention head. Furthermore, the transformation matrices are learnable, such that the model can learn to focus on specific relations between tokens in the QK^T matrix. For example, an attention head which has learned the relation between direct objectives and respective verbs can after the Softmax operation have a matrix where the column corresponding to a verb would have high values in rows corresponding to the direct objectives used with that verb. Finally, by applying the Value matrix to the Softmaxed values, the verb column will be computed as a sum of the Value vectors corresponding to the selected direct objectives. Finally, the output is projected back from the smaller d_{value} dimension to the larger d_{model} dimension. The final output vector from the attention process for the verb now contains additional contextual information about the direct objects relating to that verb. And this contextual information is added to the original vector for the verb. Multi-Head Attention can be thought of as doing this process of extracting contextual information with respect to different aspects of the tokens, and then adding the contextual information from each head to the original vector afterwards.

The division by $\sqrt{d_{key}}$ is a scale factor added for numerical stability when running backpropagation through the attention function. In [51], the Key and Value is always transformed from the same embedding, while the Query can differ depending on the situation, which is shown in the next section.

2.8 The Transformer Architecture

The Transformer is the underlying architecture used in almost all LLMs, and have been shown to generalize better than specific-purpose models across a variety of tasks [6, 41]. In this section the transformer architecture will be introduced in detail, covering the encoder and decoder blocks in detail as defined by [51]. And finally, connecting the components together and presenting the entire transformer architecture.

The transformer [51] consist of an encoder and a decoder. The encoder converts embedded tokens from the input into a vectorized representation that can be understood by the decoder, and the decoder uses the encoded tokens outputted from the encoder together with previously generated tokens predict what tokens should come next in the sentence it is generating. The input text is first tokenized using BPE, and the tokens are then converted into learned vector embeddings. At this stage, each vector is of size $d_{\text{embedding}}$, which will be refered to as d_{model}. The learned embeddings are shared for both the encoder and decoder. Furthermore, a positional encoding is added to the embedding, which is necessary because the attention mechanism does not account for order of its inputs, and the attention layers are the only layers in which the model communicates between tokens.

2.8.1 Encoder

The embedded representation of the input tokens are passed to the encoder where multi-head self-attention is applied followed by a feed-forward network. There are also residual connections and layer-normalization applied after both the self-attention block and feed-forward block. The reasoning behind having a residual connection in the attention block is explained in Section 2.7. Before defining the encoder block mathematically, some definitions must be introduced:

The embedded and positionally encoded input vectors to the encoder are denoted as $\mathbf{enc}_{\text{in}} \in \mathbb{R}^{T_{\text{enc}} \times d_{\text{model}}}$, where T_{enc} is the number of tokens in the input to the encoder and d_{model} is the dimensionality of each token embedding.

Let MultiHead(Q, K, V, M, h) be defined as in Equation 2.6.

The Layer Normalization operation is defined as:

$$\text{LayerNorm}(x) = \gamma \cdot \left(\frac{x - \mu}{\sqrt{\sigma^2 + \epsilon}} \right) + \beta \tag{2.8}$$

where $\gamma, \beta \in \mathbb{R}^{d_{\text{model}}}$ are learnable parameters, $\epsilon \ll 1$ ensures numerical stability, and $\mu, \sigma \in \mathbb{R}^{T_{\text{enc}}}$ are the mean and standard deviation computed across the d_{model} dimension of the input x. The LayerNorm normalizes each vector embedding without any cross-talk between the embeddings.

The encoder block operations are defined as follows:

$$\mathbf{enc}_1 = \text{LayerNorm}\left(\mathbf{enc}_{\text{in}} + \text{MultiHead}\left(\mathbf{enc}_{\text{in}}, \mathbf{enc}_{\text{in}}, \mathbf{enc}_{\text{in}}, \mathbf{0}, h\right)\right) \tag{2.9}$$

$$\mathbf{enc}_{\text{out}} = \text{LayerNorm}\left(\mathbf{enc}_1 + \text{FFN}\left(\mathbf{enc}_1\right)\right) \tag{2.10}$$

where FFN$(\cdot)$ denotes the feed-forward network defined as ReLU$(0, xW_1 + b_1)W_2 + b_2$, with weight matrices $W_1 \in \mathbb{R}^{d_{\text{model}} \times d_{ff}}$, $W_2 \in \mathbb{R}^{d_{ff} \times d_{\text{model}}}$, and bias vectors $b_1 \in \mathbb{R}^{d_{ff}}$, $b_2 \in \mathbb{R}^{d_{\text{model}}}$ being learnable parameters.

2.8.2 Decoder

The decoder uses the same learned embeddings as input, and apply multi-head self-attention with a non-zero mask in order to only allow token i to communicate with tokens $< i$. After the masked multi-head self-attention is applied, multi-head attention is performed using the encoder output as Key and Query and the computed values in the decoder as Value. After this, the decoder uses a feed-forward network, similar to the encoder. Residual connections and layer normalization is performed on each block in the decoder.

Let $\mathbf{dec}_{\text{in}} \in \mathbb{R}^{T_{\text{dec}} \times d_{\text{model}}}$ be the inputs to the decoder. T_{enc} is the number of generated tokens and d_{model} is the dimensionality of each token embedding.

Let $M \in \mathcal{R}^{T_{\text{dec}} \times T_{\text{dec}}}$ be a mask matrix that removes the ability to have connections with tokens later in the sentence, which preserves the auto-regressive property of the model which also makes training the model a lot faster. T_{dec} is the number of tokens that is currently used in the decoder. The mask matrix is lower-triangular:

$$M = \begin{pmatrix} 0 & -\infty & \cdots & -\infty \\ 0 & 0 & \cdots & -\infty \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 0 \end{pmatrix} \tag{2.11}$$

Using the defined functions, the decoder is defined by:

$$\mathbf{dec}_1 = \text{LayerNorm}(\mathbf{dec}_{\text{in}} + \text{MultiHead}(\mathbf{dec}_{\text{in}}, \mathbf{dec}_{\text{in}}, \mathbf{dec}_{\text{in}}, M, h)) \tag{2.12}$$
$$\mathbf{dec}_2 = \text{LayerNorm}(\mathbf{dec}_1 + \text{MultiHead}(\mathbf{dec}_1, \mathbf{enc}_{\text{out}}, \mathbf{enc}_{\text{out}}, \mathbf{0}, h)) \tag{2.13}$$
$$\mathbf{dec}_3 = \text{LayerNorm}(\mathbf{dec}_2 + \text{FFN}(\mathbf{dec}_2)) \tag{2.14}$$

where $\text{FFN}(\cdot)$ is defined as in the encoder.

2.8.3 End-to-End Functionality of the Transformer Architecture

Finally, the encoder and decoder blocks are stacked $N = 6$ times[51]. The encoder blocks are sequentially connected by feeding enc_{out} from the one block as input to the next block. The encoder output from the last block is

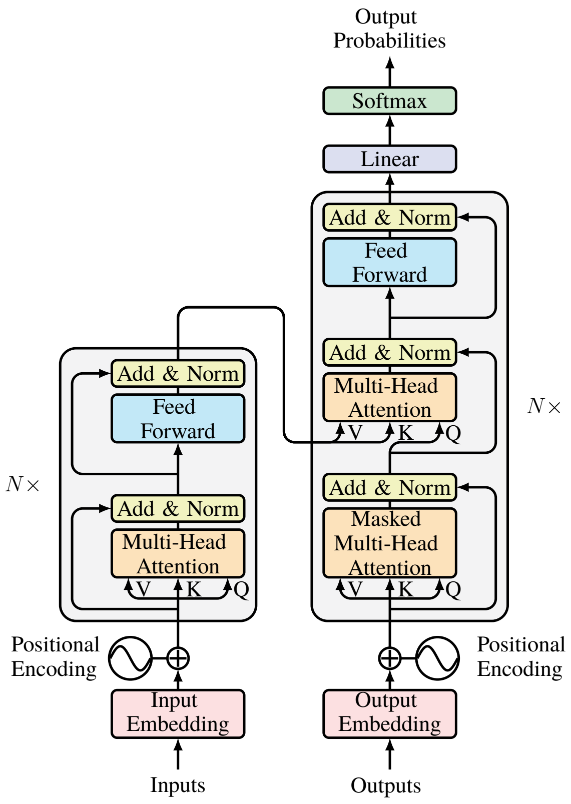

Figure 2.1: The Transformer architecture, recreated based on [51], with added clarity of which parameters that are passed to the Multi-Head attention blocks. The figure shows the encoder on the left and the decoder on the right. The Input Embedding and Output Embedding blocks use the same word embeddings.

fed into each decoder block, and the decoder blocks are similarly connected sequentially by feeding dec_{out} as input to the next deoder block. The output from the final decoder is followed by a linear layer that combines the output and change the dimensionality to be equal to the token size, producing logits as output. Finally, Softmax is applied to transform the logits into a probability distribution over how confident the model is in its predictions. The next token is predicted based on the output vector for the last token in the sentence, which only require the output vector for the last token to be processed by the linear layer and Softmax operation. The entire transformer architecture is shown in Figure 2.1.

2.9 Transfer Learning

Training LLMs from scratch is a compute intensive task that may take months on large GPU-clusters and require terabytes of training data [58]. Hence, some datasets are too small to use when training LLMs from scratch. In this case, it is instead better to fine-tune an existing pre-trained model.

Transfer learning is a method to increase the performance in a target domain by transferring knowledge from a related source domain, on which the model has been pre-trained on before being Fine-Tuned (FT) on the target domain [59]. The idea builds upon the target and source domain sharing common features. For example, if one knows how to ride a bike it will be easier to learn to ride a motorcycle, as both share common features such as balancing on two wheels. And if one knows how to read and has a general understanding of what words mean, it will be easier to learn about a specific topic in that language.

In the domain of LLMs, it is common to fine-tune models for specific tasks such as chatting [6], generating code in a programming language [60] and more.

2.10 Efficient Fine-Tuning

While transfer learning solves the problem of learning from little data, there also exist barriers with respect to computational cost and memory. Language models using transformers can be very large, and fine-tuning a model requires just as much memory as pre-training it, most of the memory being a result of backpropagation and keeping track of extra values in the optimizer. For example, the GPT3 model fits in 350GB of VRAM but require at least 1.2TB of VRAM during training when using the Adam optimizer [61]. To make fine-tuning more efficient in terms of memory and cost, efficient fine-tuning methods have been developed. One way to decrease memory usage and increase speed is to decrease the floating point precision of weights, allowing each weight to take up fewer bits in memory and computations to be quicker. Another method is to only fine-tune some layers of the network, to avoid the memory overhead of keeping track of optimizer values for some weights. But there also exist more sophisticated methods, which will be covered in this section. PEFT includes methods which fine-tune few parameters in comparison to the model size, while still updating parameters for all layers.

2.10.1 Adapter models

One way to avoid fine-tuning the entire model is to add new layers in between existing layers in the model, and train those instead. The Adapter [62] is a bottleneck feed-forward network, with a down-projection from the model dimensionality to a lower dimensionality followed by an up-projection back to the model dimensionality again. It also has a residual connection. The Adapter block is inserted into transformer blocks before the Add & Norm is performed in Figure 2.1. During fine-tuning, only the Adapter weights are trained and the rest of the model is frozen. In total, only a few percent of the parameters in the final model is made up of the Adapter weights, making training much less memory expensive compared to fine-tuning a full model. Despite the decrease in computation, [62] show that the FT models using Adapters perform within 1 percent point compared to fine-tuning the full model on the GLUE [63] dataset.

AdaMix [64] takes an approach similar to Sparse Mixture of Experts (SMoE) in the adapters, but the expert selection is random at each iteration instead of using a gating function. The model tested by [64] uses 4 adapters for down-projection and 4 for up-projection during fine-tuning, where both projections are selected at random in each iteration. After fine-tuning, the experts are merged into one by averaging the weights of the experts. This method outperforms full model fine-tuning by 1 percent point on GLUE.

2.10.2 Low-Rank Adaptation Fine-Tuning

Low-Rank Adaptation (LoRA) fine-tuning [61] builds on the fact that weights of trained LLMs has been shown to be of low rank, and can therefore be compressed into lower dimensionality [65]. The matrices are assumed to be of a low rank, r, indicating that a weight matrix can be factorized as $W = AB^T$, where $W \in \mathcal{R}^{d \times k}$ and $A, B \in \mathcal{R}^{d \times r}$, where r is the assumed rank of W. During fine-tuning, it should therefore be sufficient to update the weight matrices A and B, instead of using W directly, which reduces the number of required parameters. However, projecting W into a lower dimension will always cause some loss of information. Therefore, a hybrid method is introduced, where the pre-trained weight matrix, W_0, is frozen and new parameters A, B are initialized and used in fine-tuning. The forward-equation for a matrix multiplication using LoRA is done as $W_0x + BA^Tx$, where A and B are learnable matrices, and W_0 is the pre-trained matrix and is never updated. The assumed rank, r, which determines the size of A

and B, is a selectable hyperparameter which can be interpreted as the rank of the difference between the pre-trained matrix and the optimal FT matrix, $BA^T \sim W_0 - W_{optimal}$ acting as the two factors of a low-rank fine-tuneable matrix which is added to the original pre-trained weight.

Using LoRA reduces the number of gradients to be updated, which decreases the memory usage for gradients and optimizer values. Furthermore, [61] only fine-tune the attention and do not train the MLP modules in their tests, but note that the method can be generalized to any neural network architecture. It is also found that $r = 1$ is sufficient for good performance and performs similarly to fine-tuning the entire network across benchmarks, including the Text-to-SQL benchmark WikiSQL.

QLoRA [66] is a method that builds upon LoRA, but further optimize memory usage by using quantization. Quantization is a method which can reduce the size of floating point numbers, and hence the size of the weights in the model. In QLoRA the original model weights are changed to 4-bit NormalFloat instead of the 16-bit BrainFloat that are used in most models, hence decreasing the size the model takes up in memory by a factor of 4.

NormalFloat normalize values into the range $[-1, 1]$ such that the largest normalized value is 1 or -1 and then use bins distributed according to a normal distribution, where bins are thinner near 0 and wider near -1 and 1. The normalized values are then stored according to the bin in which they belong. The NormalFloat method is efficient because trained weights are mostly [66] distributed according to a zero-centered normal distribution, which allows for the distribution of weights across the bins to be equal, maximizing the theoretically optimal 4-bit compression given the weights. However, the normalization factor must also be stored for each group of weights that are normalized, making the method use slightly more than 4 bits per value. To perform computations with the NormalFloat weights, the compressed values can be decompressed using the normalization factor.

While the original model uses 4-bit precision, the A and B matrices are of 16-bit precision, allowing to learn precise weights while keeping the memory usage lower by quantizing the full model. Furthermore, the QLoRA setup is used in all layers, compared to only the attention layers in LoRA [61]. QLoRA also optimize for memory by paging optimizer values on the regular Random Access Memory (RAM) when the GPU VRAM is full in order to avoid out of memory if there are short memory spikes.

2.10.3 Few-Shot Prompting

Another way to teach LLMs is through in-context learning, which does not require any fine-tuning. FS in-context learning is a method which provides examples to the language model during inference [67, 68]. This technique is used to allow pre-trained general-purpose language models to excel in specific tasks, such as Text-to-SQL [4, 5, 7, 33, 34, 37]. However, there are some difficulties with using FS. For example, the ordering of the examples [68] and the selection of examples [67] is important for the quality of the generated output.

2.11 Sparse Mixture of Experts

Another method to decrease memory requirements and increase inference speed without any cost of performance is to use a SMoE architecture. SMoE is a method where multiple expert networks are trained, but only one or a few are used during inference. The selection of which network that should be used can be done at random or through a gating layer, which selects the experts that are best suited for the task. SMoE also includes models in which only a part of the model uses a Mixture of Experts (MoE) structure. For example, in transformer models, it is common that SMoE is applied only to the feed-forward network, but not to the multi-head attention [69]. When applied to the feed-forward network of a transformer, the FFN$(\cdot)$ function in Equation 2.10 and Equation 2.14 is replaced by GatedFFN$(\cdot)$:

$$\text{GatedFFN}(x) = \sum_{i=0}^{N-1} G(x)_i \text{FFN}_i(x) \tag{2.15}$$

$$G(x) = \text{Softmax}(\text{TopK}(x \cdot W_g)) \tag{2.16}$$

where $G(\cdot)$ is the gating function, FFN$_i(\cdot)$ are the experts, N the total number of experts and $W_g \in \mathbb{R}^{a \times b}$ a learnable weight matrix. The N feed-forward networks are completely disconnected and does not share any parameters, but the structure is exactly the same in each expert. TopK$(\cdot)$ selects the K largest values computed, and sets the rest to $-\infty$, making the Softmax output non-zero values only for K experts. The TopK function is what makes the MoE method sparse, by not using all N experts at once.

To train the gating layer G, backpropagation is used just like in any neural network model, training the gating layer at the same time as the rest of the model [70]. However, training a SMoE model comes with challenges, [69] finds that SMoE models are more prone to diverging loss, and most papers reviewed by [69] had some issues with loss instability. An attempt to mitigate the instability is to add a balancing loss term that penalizes large logits, $x \cdot W_g$, such as the router z-loss [71], reducing floating point errors and hence increasing stability. A load balancing loss designed to encourage the model to use all experts evenly is another attempt to solve the problem [71].

The Mixture of Experts architecture indicates that each expert is specialized at some tasks. However, it is found that the routing of tokens is decided early in training [72]. And after training the same token is always routed to the same experts regardless of the context of that token. For example, `left` has a different meaning in `Nothing left` and `Turn left`, but this context does not change the selected expert, although the embedding after each layer's multi-head attention is affected by the context. On the other hand, it has also been shown that the context of a token can change the expert it is routed to [73]. Furthermore, there is no strong correlation between experts and domains, but there is some correlation between experts and language [72]. For example, `has`, `had` and `have` are routed to the same expert [72].

Although the mentioned training difficulties, SMoE models have been shown to work well with LLMs [72, 73]. The main upside of SMoE models is that inference is compute-efficient compared to dense models with a similar number of parameters. This is due to the TopK in the gating function which sets most elements to 0, and the experts only have to be computed for the non-zeroed elements in the gating function. It has also been shown that it is possible to use parameter offloading to run SMoE with less VRAM compared to dense models of similar size [74].

2.12 Evaluation metrics

There are a few different ways to measure how good a model is at the Text-to-SQL task. The most commonly used metrics are presented in this section.

2.12.1 Component Matching

Component Matching [75] divides the SQL query into different components: `SELECT`, `WHERE`, `GROUP BY`, `ORDER BY` and `KEYWORDS`. `KEYWORDS` include all SQL keywords except column names and operators. Operators are for example $>=$ and $+$. The components that do not have order constraints, such as the items in the `SELECT` statement, are compared without respect to order. Each component in the prediction and ground truth are compared individually for an exact match, or exact set match depending on the order constraint. Hence, each component score is either 0 or 1. Finally, the F1-Score [76, 77] of the components are computed to find the component matching score for an SQL query. The score for each prediction and gold pair is thus bound in the range $[0, 1]$. To compute the Component Matching score for an entire set of values, the average component match score across the values is used.

The precision and recall used to compute the F1-Score can be interpreted as the fraction of components in the predicted query which has a component score of 1 and the fraction of components in the gold query which has a component score of 1, respectively. Since the F1-Score is the harmonic mean of precision and recall, it balances the trade-off between these two metrics, ensuring that both metrics must be high to get a good F1-Score.

2.12.2 Exact Matching

Exact Matching Accuracy (EM) [75] uses the same method as Component Matching to decompose the SQL query into components. However, the entire decomposed query must match the ground truth query to be considered as correct. This makes EM a binary metric for each datapoint, unlike component matching that considers partially matching queries as partially correct. To compute the EM score for an entire set of values, the average is used.

2.12.3 Execution Accuracy

Execution Accuracy (EX) [75] compares the return values of the execution of the predicted and gold query. The returned rows are considered as a set, such that the order of rows can be different for the predicted and gold query. Formally, the EX is be defined as

$$\mathrm{EX}(V, \hat{V}) = \frac{\sum_{n=1}^{N} \mathbb{1}_{V_n = \hat{V}_n}}{N} \tag{2.17}$$

where V_n is the set of rows produced from the gold query and $\hat{V}_n$ is the set of rows produced by the predicted query for the nth datapoint [3].

2.12.4 Valid Efficiency Score

Valid Efficiency Score (VES) [3] is a combination of EX and relative efficiency between the predicted and gold query. Efficiency can refer to execution time, memory cost, or throughput. However, in [3] they only use efficiency in terms of execution time. Therefore, VES will be explained in terms of execution time.

The score compares the execution time of the predicted SQL query against the gold SQL query and weighs the EX according to the time efficiency score. The formal definition of VES is:

$$\text{VES}(Y, \hat{Y}) = \frac{1}{N} \sum_{n=1}^{N} \mathbb{1}_{V_n = \hat{V}_n} \cdot \text{R}(Y_n, \hat{Y}_n) \tag{2.18}$$

$$\text{R}(Y_n, \hat{Y}_n) = \sqrt{\frac{E(Y_n)}{E(\hat{Y}_n)}} \tag{2.19}$$

where N is the number of datapoints, V_n and $\hat{V}_n$ are defined the same way as in Section 2.12.3. Y_n and $\hat{Y}_n$ represent the gold and predicted query, respectively. $E(\cdot)$ is a function to measure the execution time. The square root function is applied to dampen the effect of outliers as $|\sqrt{x} - 1| \leq |x - 1|$, moving any value closer to 1 than before applying the square root. The R-function is not upper-bounded as the predicted query can execute faster than the ground truth. Consequently, the overall VES score is not upper-bounded either. However, assuming that the gold queries are optimally designed in terms of execution speed, the VES score is upper bounded by 1 as the R-function must be upper bounded by 1 when Y_n is optimal.

To compute $E(\cdot)$ the execution time is estimated by executing the same query multiple times. In [3], the SQL query is executed 100 times and the VES score for N samples are computed according to Algorithm 1, where `remove_outliers` removes any value which is more than three standard deviations away from the mean of the 100 generated samples and `execute_sql` executes the SQL and computes the time the execution takes.

Algorithm 1 Valid Efficiency Score

```
1: function VES($V, \hat{V}, Y, \hat{Y}$)
2:     $VES \leftarrow 0$
3:     for $n \leftarrow 1$ to $N$ do
4:         if $V_n = \hat{V}_n$ then
5:             differences $\leftarrow \{\}$
6:             for $i \leftarrow 1$ to 100 do
7:                 predicted_time $\leftarrow$ execute_sql$(\hat{Y}_n)$
8:                 gold_time $\leftarrow$ execute_sql$(Y_n)$
9:                 time_fraction $\leftarrow$ predicted_time/gold_time
10:                append time_fraction to differences
11:            differences $\leftarrow$ remove_outliers(differences)
12:            $R\_score \leftarrow \sqrt{\frac{\sum(\text{differences})}{\text{length}(\text{differences})}}$
13:            $VES \leftarrow VES + R\_score$
14:    $VES \leftarrow \frac{VES}{N}$
15:    return $VES$
```

2.13 Datasets

There exist quite a few Text-to-SQL datasets [3, 15, 75, 78, 79, 80, 81, 82, 83, 84, 85]. However, only two datasets are widely used for benchmarking LLMs. These datasets are Spider [75] and BIRD [3]. The datasets consist of databases, as well as written natural language questions called utterances paired with ground truth SQL queries called gold queries. A comparison of the sizes of the Spider and BIRD datasets is shown in Table 2.2. Both datasets uses the SQLite database language for the databases.

Table 2.2: A comparison of the contents in the Spider and BIRD datasets.

Metric	**Spider**	**BIRD**
Databases (train / val / test)	206 (146 / 20 / 40)	95 (69 / 11 / 15)
Tables / Database	5.1	7.3
Rows / Database	2 K	549 K
Datapoints (train / val / test)	11 840 (8 659 / 1 034 / 2 147)	12 751 (9 428 / 1 534 / 1 789)

2.13.1 Spider

Spider [75] was the first large-scale cross-domain Text-to-SQL dataset. The databases in the dataset were first collected and then manually altered such that the naming was semantically understandable. The collected databases come from college database courses, online SQL tutorials, Databaseanswers* and 90 databases from the WikiSQL [15] dataset. The databases were then populated with fabricated data. After the collection process, questions were created manually, and SQL queries were created to answer the questions, yielding a total of 10 181 questions answered with 5 693 SQL queries. However, in the released version of Spider†, six existing datasets are also incorporated in the Spider dataset. The combination of these six datasets and the data created by [75] is displayed in Table 2.2.

Spider uses the EM and EX scores to evaluate performance of models on their dataset.

2.13.2 BIRD

Big Bench for Large-scale Database Grounded in Text-to-SQL Tasks (BIRD) [3] is designed using real-world data gathered from Kaggle, CTU Prague Relational Learning Repository, and by fetching and designing databases from other accessible data. The naming from the fetched databases is kept as-is. However, names with abbreviations are given a non-abbreviated description which can be accessed in the dataset but is not used in the database itself. The database sizes are larger than other Text-to-SQL datasets, making writing efficient SQL queries of importance. Therefore, the VES score is introduced and used as a metric of how fast the generated SQL queries are executed compared to the gold queries. The other metric used to benchmark models on BIRD is EX.

Each data point consists of an utterance, a gold SQL query, and evidence. The evidence explains additional specific knowledge regarding the database required to answer the question, for example, providing information that is not clear by reviewing the database schema, such as if entries in a column use abbreviations or not, in which case the relevant abbreviation is provided. Example utterances with evidence and gold query from the train dataset are shown in Table 2.3. In the first example, the `Reach` column and unique

*databases.biz
†yale-lily.github.io/spider

users are linked together via the evidence, which is a connection that would otherwise have been difficult to make without prior knowledge of the database. However, the evidence is sometimes used to provide partial SQL queries, such as in Example 2 in Table 2.3, where the average number of likes in the evidence is given in a very similar form to how it is solved in the gold query. Furthermore, each column is provided with a short description of what the name means, and in some cases a description of the values which can be expected in the column. For example, if a text column is used to enumerate data, containing only a specific set of values.

The evaluation dataset also contains a label specifying how hard it is to answer the utterance, with the three categories simple, moderate and challenging.

Table 2.3: Question, Evidence, and gold query for two training examples in the BIRD dataset

	Example 1	Example 2
Utterance	How many unique users have seen tweet with text ”Happy New Year to all those AWS instances of ours!”?	What is the average number of likes for a tweet posted by a male user on Mondays?
Evidence	”Happy New Year to all those AWS instances of ours!” is the text; seen unique users refers to Reach	male user refers to Gender = ’Male’; ’Monday’ is the Weekday; average number of likes = Divide (Sum(Likes), Count(TweetID))
Gold query	`SELECT Reach FROM twitter WHERE text = "Happy New Year to all those AWS instances of ours!"`	`SELECT SUM(T1.Likes) / COUNT(T1.TweetID) FROM twitter AS T1 INNER JOIN user AS T2 ON T1.UserID = T2.UserID WHERE T2.Gender = 'Male' AND T1.Weekday = 'Monday'`

The test dataset used in BIRD is hidden, but models can be evaluated by contacting the authors for submission[*]. The motivation behind keeping the test set private is to avoid data leakage. The test set databases do not come from the public databases used as training and evaluation data, but are instead designed from scratch to avoid data contamination, for example by appearing in the pre-training dataset of LLMs.

[*]bird-bench.github.io

Chapter 3

Related Work

In this section, the recent advancements in Text-to-SQL is presented. Firstly, the relevant research papers are introduced and compared, based on the prompting strategies used and the output generation techniques used. And finally, the overall findings are summarized and the models are compared in terms of performance on both Spider and BIRD.

3.1 Prompting Strategies

Various prompting strategies are used, but the majority of the LLM based methods use a format which start with some instructions, followed by the knowledge representation as text and utterance. In the GPT-3.5 based methods, ChatGPT-SQL [86] and C3 [38], a "clear prompting layout" is used, where the instruction, knowledge representation and utterance are split by ###, in an attempt to help the model separate the sections in the input. The other methods, which use GPT-4, also have a delimiter between each section, although it is not explicitly mentioned in their work. The GPT-4 based method DIN-SQL [7] focus on designing different prompts depending on how difficult the utterance is to answer. The utterance is classified into three categories depending on the complexity of the SQL to be generated:

- Easy: Question, schema link, and answer format with FS. And the final question is asked in the same format, but with the answer missing.
- Non-Nested Complex: The answer format is changed compared to the easy prompt. It uses NatSQL [87], an intermediate representation between natural language and SQL developed for Text-to-SQL. Both

NatSQL and SQL answers are provided. Furthermore, the input prompt ends with `Let's think step by step`, to encourage the model to reason before answering. After the reasoning, a NatSQL answer and an SQL answer is written. This format is used for FS examples, and the final question has the same structure but ending at `Let's think step by step`.

- Nested Complex: Starts by identifying and solving the sub-queries required, the general prompting layout is otherwise the same as for non-nested complex. But in the reasoning step before producing the final SQL, the sub-queries are also addressed.

Using different prompts depending on the difficulty of the utterance is unique to DIN-SQL.

The currently best model on the BIRD test dataset is MAC-SQL [4], which first apply schema filtering, and then uses a prompt which asks to decompose the utterance as a list of questions, where each question can be answered with only one `SELECT` statement and using the answers from questions earlier in the list. To achieve this, a CoT prompting strategy is used as well as providing FS examples. The final prompt includes an instruction and constraints, mitigating the bias of the model, database information, and the utterance, and the model generates the questions and sub-queries within the same answer. Similar to DIN-SQL, MAC-SQL also generates sub-queries before producing the final SQL query.

DAIL-SQL [5] compares various FS prompting strategies, and unlike the previous methods, they use Retrieval Augmented Prompting (RAG) to select the FS examples, based on both utterance similarity and query similarity against a database of existing datapoints. The query similarity is computed by first generating a query for the utterance using the Graphix-T5 [23] model, and then comparing that query against the gold query. For both FS examples and the final prompts, various prompting strategies are tested. For the FS examples, three prompts are tested:

- Full information prompt, containing the database schema, utterance, and gold SQL.
- SQL-Only prompt, containing only the SQL queries.
- DAIL organization prompt, containing the SQL query and utterances, but not the database schemas.

and for the final prompt to be answered, three different prompts are tested:

- Code representation prompt: Entire database schema is included in DDL format.
- OpenAI demonstration prompt: Uses compact notation with no key information and the utterance placed after the database information.
- Alpaca SFT Prompt: Similar to the OpenAI demonstration prompt, but the utterance is placed before the database information.

The best combination of prompts is found to be a code representatin prompt together with the DAIL organization prompt for the FS examples.

Another method which uses RAG manage to outperform many of the methods using GPT-4 [34]. In SQL-PaLM [37], FS and FT is compared using the PaLM-2[88] model, and it is found that there is no big difference in performance between the two methods.

In summary, many of the prompting techniques use schema-filtering, and some use Zero-Shot, while others use FS, and it has been shown that FS can increase the EX on Spider and BIRD development dataset by 8 and 5 percent, respectively. However, the zero-shot methods are a lot cheaper and can also be faster as they do not use as many tokens. Not many have tried to use FT models, and with PaLM-2, no increase in performance was shown when Fine-Tuning the model.

3.1.1 Output Generation

The output generation techniques mainly use either Self-Correction or Self-Consistency methods. However, DAIL-SQL [5] use the generated output from GPT-4 without modifying it, and outperforms most other techniques in terms of EX.

MAC-SQL [4] uses Self-Correction, and prompt GPT-4 until the generated SQL no longer contain errors. In [34], a “Dynamic Revision Chain” is used, updating the query with feedback regarding error as well as a natural language explanation of the generated query. The query is updated iteratively until the model generates the same query twice, or until a fixed cutoff of N steps is reached. It is found that using $N = 2$ is optimal, increasing EX on Spider by 5 percent points compared to not using the revision chain. In contrast to these methods, DIN-SQL also uses Self-Correction, but finds that it is not necessary to provide the SQL errors to GPT-4, but rather only provide guidance concerning standard issues that SQL generation may contain, such as using `DISTINCT` keyword when needed. However, DIN-SQL also performs worse than the methods providing the error as feedback.

Both SQL-PaLM [37] and C3 [38] use Self-Consistency, generating multiple outputs for the same prompt and selecting the most commonly generated prompt. In [38] it is found that the Self-Consistency method adds 1.3 percentage points to the performance on Spider, compared to the 5 percentage point increase using the "Dynamic Revision Chain" for Self-Correction.

3.2 Literature Summary

All recent papers use LLMs, most by developing new prompting strategies rather than fine-tuning the model. The only exception is SQL-PaLM, which fine-tunes PaLM-2 [88] and finds that the FT model has a similar performance to the pre-trained model. The papers with reported results on BIRD and Spider are presented in Table 3.1 and Table 3.2, respectively. Each paper tries different prompting strategies. However, some of the most used strategies are schema linking and FS prompting. The general prompt layout follows the order of [Instruction] [Schema] [Utterance], with a variety of schema formatting and instructions as well as spacing and formatting of the layout. The proprietary models, such as GPT-4 and ChatGPT are continuously improved, and consequently, the results for a method tested on new iterations of the models may achieve better results only due to the fact that the model itself is improved, rather than due to using better prompting strategies and output generation techniques.

Table 3.1: Performance metrics for models evaluated on the BIRD dataset, with details on knowledge representation and output generation methods. The metrics are reported according to the official benchmark scores at bird-bench.github.io.

Model	Knowledge Representation	Output Method	EX ↑		VES ↑	
			Dev	Test	Dev	Test
DIN-SQL+GPT-4 [7]	Text	Self-Correction	50.72	55.90	58.79	59.44
DAIL-SQL+GPT-4 [5]	Text	None	54.76	57.41	56.08	61.95
MAC-SQL+GPT-4 [4]	Text	Self-Correction	57.56	59.59	58.76	67.68

Table 3.2: Performance metrics for models evaluated on the Spider dataset, with details on knowledge representation and output generation methods. Scores are reported according to the official benchmark scores at yale-lily.github.io/spider. Entries marked with * indicates that the score is taken from the paper and not the official benchmark, and is only used when the score is missing in the official benchmark. Entries marked with - indicates that the score is not reported.

Model	Knowledge Representation	Output Method	EM ↑		EX ↑	
			Dev	Test	Dev	Test
Graphix-T5 [23]	Relational	PICARD	77.1	74.0	81.0*	77.6
ChatGPT-SQL [86]	Text	None	-	-	70.1*	-
DIN-SQL+GPT-4 [7]	Text	Self-Correction	60.1	60	74.2*	85.3
SQL-PaLM FT [37]	Text	Self-Consistency	-	-	82.8*	-
SQL-PaLM FS [37]	Text	Self-Consistency	-	-	82.7*	-
C3+ChatGPT [38]	Text	Self-Consistency	-	-	81.8*	82.3
DAIL-SQL+GPT-4 [5]	Text	None	71.9*	-	82.4*	86.2
RAG+ChatGPT [34]	Text	Self-Correction	-	-	85.0*	-
MAC-SQL+GPT-4 [4]	Text	Self-Correction	63.2*	-	86.8*	82.8*

Chapter 4

Method

This chapter covers the selection of the dataset, the motivation behind the selection of LLMs for FT, and the motivation behind using Claude Opus. Then, the prompt designs used are presented, followed by the experimental setup, the method of the error analysis, and finally, the statistical methods used.

4.1 Dataset & Metrics

The datasets Spider and BIRD are both of similar size, covering a wide variety of domains and with a similar amount of columns per table. However, the BIRD dataset contains a lot more data per database, and the gold queries are designed to be efficient. Furthermore, it uses a wider variety of SQL keywords, such as functions. Therefore, the BIRD dataset was chosen. And consequently, the evaluation metrics chosen were EX and VES to obtain comparable results with prior work, as these are used to benchmark BIRD. However, when analyzing and comparing models, only the EX score is used. As the training and evaluation data were publicly available, these splits were used for training and evaluating the model.

4.2 Selection of LLMs

Both a large pre-trained model and a smaller model to fine-tune was selected, the smaller model was limited in size such that it could be trained on an 80GB VRAM A100 GPU. And the large pre-trained model was used via an API,

Table 4.1: Performance of closed source LLMs on the datasets MMLU and HumanEval. The scores are provided by the respective API providers.

Model	MMLU (5-shot)	HumanEval (0-Shot)	Context Length	Cost (I,O/Mtok) $
GPT-4	86.4 %	67.0 %	128K	10$ / 30$
Claude Opus	86.8 %	84.9 %	200K	15$ / 75$
Mistral-Large	81.2 %	45.1 %	32K	8$ / 24$
Gemini 1.5 Pro	81.9 %	71.9 %	1M	7$ / 21$

with a cost per input and output token.

4.2.1 Large Pre-trained Model

When selecting the closed source model, the most important factors were the performance and generalization of the model. The model should have a good knowledge of SQL, and not require further fine-tuning to be used. In Chapter 3, the two most popular closed source models were GPT-3.5 and GPT-4, where the GPT-4 model outperforms GPT-3.5. A variety of highly capable models are available through APIs from different companies. These include OpenAIs GPT-4, Googles Gemini Pro, Antrhopics Claude 3 Opus, and Mistrals Mistral-Large. Neither of these models has released details about the specific model architecture, but all models are Transformer-based. The performance on the MMLU [89] and HumanEval [90] datasets are used to compare the models before selection. MMLU covers knowledge across 57 different domains which require a good problem solving ability and world knowledge. HumanEval is a dataset designed to test the coding abilities of models. A comparison of the models on MMLU and HumanEval is shown in Table 4.1, where Claude Opus has a notably higher score than the other models on HumanEval, indicating that the model is good at coding compared to the other models. Furthermore, both GPT-4 and Claude Opus were used in preliminary experiments on the training data, where both models showed similar performance on the tested training data. I chose to use Claude Opus, as all prior work on BIRD use GPT-4. The Claude Opus version used was `claude-3-opus-20240229`.

4.2.2 Smaller Fine-tuneable Model

As the model will be FT, the weights of the pre-trained model must be open source. Furthermore, the model must also be available for commercial use, as the final model may be used commercially. There exist a lot of pre-trained models which are available to download and free to use commercially. Some models meeting these conditions are Mistral, Mixtral, Llama, Llama-2, and Phi-1. Each model introduces some novelty to get as much performance out of the model as possible. These modifications range from architectural modifications to using high-quality data to using SMoE. However, all models have in common that they use the decoder-only transformer architecture.

The Llama [91] and Llama2 [92] models use normalization before the attention block and before the FFN block, instead of after. Furthermore, SwiGLU [93] is used as the activation function and Rotary Embeddings [54] are used instead of the positional embeddings used in [51]. Both Llama and Llama2 exist in different sizes, ranging from 7B to 70B parameters. The data used is open-source, with the majority (67 %) coming from the Common Crawl[*] dataset. In total, the models are trained on 1-1.4 trillion tokens, with 1T for the smaller models and 1.4T for the larger models. There also exist a few studies on fine-tuning Llama models [66, 94]. Orca [94] experiments with using GPT-4 augmented data to fine-tune the Llama model. The data is created by prompting GPT-4 to use reasoning before answering questions in the prompt, which allows the Orca model to mimic the reasoning of GPT-4. [66] introduces QLoRA and uses it to fine-tune the Llama models on the OASST1[†] dataset, creating the Guanaco model. Furthermore, it has been shown that using a small high-quality dataset outperforms using a large dataset with worse quality [66].

The Phi-1 [95] model is a 1.3B model specified to learn Python programming. The focus of Phi-1 is to show the importance of high-quality data, training the model with only 7B tokens, compared to Llama which used 1T tokens. The data is a combination of synthetic data generated by GPT-3.5 and data from The Stack and StackOverflow, which was filtered to only include code with educational value. The FT Phi-1 model outperforms GPT-3.5 on the HumanEval benchmark, despite being more than 100 times smaller than GPT-3.5.

Mistral [96] is a 7B model using sliding-window attention, restricting the mask matrix M in Equation 2.11 such that a token only can see the k previous tokens in the input text, instead of being able to attend with all previous tokens. This is cheaper to compute but at the cost of restricting the attention window. However, it is noted that the effective attention window size increases over multiple layers. For example, the influence of token $i-k$ on token i allows

[*]https://commoncrawl.org/
[†]https://huggingface.co/datasets/OpenAssistant/oasst1

token $i + k$ in the next layer to also be influenced by token $i - k$, through token i. In general, after l attention layers, information regarding token i can influence tokens up to token $i + k \cdot l$ tokens after l layers, with a window of k tokens. Hence, using enough layers results allows for full communication across all tokens.

The Mixtral 8x7B [73] model utilizes a SMoE architecture as described in Section 2.11. The Mixtral model uses 12.9B active parameters for each token during inference, and the entire model has 46.7B parameters. The model is at least as good as GPT-3.5 and Llama 2 70B. Furthermore, on needle-in-a-haystack tests, Mixtral obtains a $100\,\%$ retrieval accuracy when tested with a 32K context length. A needle-in-a-haystack test is used to assess how well the model can attend to the entire context length by placing a key somewhere within a long text, and assessing the ability of the model to retrieve that key.

The MMLU and HumanEval scores for the fine-tuneable models are presented in Table 4.1. Although Phi-1 has the highest HumanEval score out of the model, Mixtral was considered more promising as it is not trained on only Python programming and otherwise has the best benchmark scores. Furthermore, the SMoE architecture may help the model excel at Text-to-SQL as different parts of the queries require different abilities. Therefore, the Mixtral model was chosen as the model to fine-tune. Specifically, the instruct FT version of Mixtral was used.

4.3 Prompt Design

The process of designing the prompts for Mixtral and Claude Opus was done differently. Mixtral is used for fine-tuning and therefore the Mixtral prompt was designed to be simple since the fine-tuning process will teach the model how to write correct SQL. In contrast, the prompt for Claude Opus was designed to add instructions to mitigate biases and avoid making common mistakes. Furthermore, a prompt used for Mixtral without fine-tuning was also designed to get a baseline of how good Mixtral is without fine-tuning, this prompt was designed similarly to the method used to design prompts for Claude Opus. For both models, it was decided to use the DDL format for providing the database schema, without providing the extra column information that exist for some columns by the BIRD dataset. Furthermore, it was decided to not provide example values for each column in the schema. This is mainly a restriction from the company, as the data in some columns will not be allowed to be sent to a third party API. The prompts used are presented in Appendix A.

4.3.1 Claude Opus

The prompt used for Claude Opus is inspired by the existing papers on the BIRD dataset using GPT-4. The prompt was designed to be cost-efficient, limiting the cost per prompt where necessary. Therefore, FS was not used, and schema linking was decided unnecessary as Claude Opus has a long context window and good retrieval accuracy across the entire context window. The prompt structure draws inspiration from the existing prompts used for GPT-4-based methods. The final prompt used was iteratively tested against some data in the BIRD training dataset, and the prompt was changed to fix common errors and biases found in this early testing stage.

When testing on the training data, it was also found that Claude Opus sometimes generates invalid SQL, often due to non-standard column naming conventions in the schema which are ignored, and sometimes due to other syntactical errors in the SQL. Therefore, it was decided to use a self-correction step. It was found that by simply prompting the SQL error to the model, as a continuation to a chat with the original prompt and answer, the model was able to correct these errors first try in almost all situations, therefore only one iteration of self-correction was used.

The entire prompt used for Claude Opus is presented in Section A.1.

4.3.2 Mixtral

The prompt used for FT Mixtral was designed to only contain the necessary information, allowing the fine-tuning process to learn how to generate SQL rather than attempting to mitigate these biases through prompting before fine-tuning the model. No self-correction was used as the fine-tuning process made the model learn the specific prompting format used in the questions. And there are no non-executable SQL queries to learn from, making it difficult to teach the model how to fix an incorrect SQL query. The fine-tuning prompt is available in Section A.2.

Finally, the pre-trained Mixtral baseline without fine-tuning was designed using the same method as when designing the Claude Opus prompt. However, it was remarkably harder to get the model to consider all instructions in the prompt, compared to using Claude Opus. Therefore, the self-correction step was more thorough, including more instructions and not using the chat-like template used with Claude Opus, but instead re-providing the schema, utterance, and evidence together with the error. This prompt is presented in Section A.3.

4.4 Error Analysis

To find what the models are good or bad at, an error analysis was conducted, looking at all data points considered to be false in the evaluation dataset. These data points were classified based on the types of errors that caused the datapoint to be incorrect. The categories are the following:

- **Incorrect Gold Query**: The gold query does not answer the utterance correctly.
- **Exception**: When the generated query contains a syntactical error, or other errors which cause an exception when executing the query. This also includes time-limit-exceeded errors, defined as any query taking more than 30 seconds to execute.
- **Semantic Correct**: When the generated query answers the utterance correctly, but the answer is missed by BIRDs evaluation script. This includes results that differ due to rounding errors, having different orders of the returned columns, but the correct set of columns. Furthermore, if the utterance does not explicitly ask for a specific column value to be returned, and the generated and gold query returns a correct interpretation of the utterance, it is also considered semantically correct.
- **Misunderstand**: When the LLM misunderstands the utterance, evidence, or provided schema, and therefore generates a query which does not answer the utterance. SQL queries are considered as misunderstanding the utterance or evidence if there is no attempt to use all the necessary information, and misunderstanding the schema includes misinterpreting column names and performing incorrect joins.
- **Dirty Database Values**: When the only difference between the result of the gold query and the generated query is due to null values in the database.
- **Missing Information**: When the model is not provided with enough information to be able to correctly answer the utterance. This includes if the utterance or evidence asks for a string with incorrect capitalization, if multiple table names in the schema can be interpreted as correct to

answer the utterance, or if the extra information provided in the schema description, which is not included in the DDL schema provided to the model, is required to answer the utterance.

- **Other Errors**: When the model understands the utterance, evidence, and schema, but fails to answer the utterance due to issues such as incorrect ordering, or a lack of SQL knowledge. Furthermore, blatantly wrong queries, for example, a query that ignores the utterance completely. The reason for assigning such queries to other errors instead of the Misunderstand category is that the errors most likely do not stem from incorrectly understanding the utterance, evidence, or schema but rather neglecting the input completely. The difference between a "Misunderstand" and these errors is that in a "Misunderstand", some necessary information is used, such as using the correct tables or considering some correct conditions.

Examples of each category from the BIRD evaluation dataset are provided in Appendix B.

4.5 Experimental Setup

The experimental setup covers the software and hardware used, as well as provides details on which methods were used to fine-tune Mixtral and how the evaluation of Claude Opus and Mixtral was done.

4.5.1 Software

The framework used for fine-tuning and inference with Mixtral was Axolotl*, which is a framework built on top of HuggingFace to easily fine-tune LLMs. The framework supports both LoRA and QLoRA, and it allows to specify the dataset format and all necessary hyperparameters. The Axolotl framework uses Python.

Claude Opus is accessed through an API using Python.

The evaluation code provided by BIRD is also implemented in Python, and adapted to be used with in my code to evaluate SQL queries.

The statistical methods presented in Section 4.6 used the Python libraries Scipy and Numpy. Numpy was used to compute the confidence intervals using

*github.com/OpenAccess-AI-Collective/axolotl

Equation 4.1. P-values were computed using `scipy.stats.t.sf` with T and df computed via Equation 4.2 and Equation 4.4.

4.5.2 Hardware

To fine-tune Mixtral, the service Paperspace was used, where a machine with an A100 GPU with 80GB VRAM was used for both training and inference. The cost of this machine was covered by the company.

4.5.3 Fine-Tuning

In order to be able to fine-tune Mixtral 8x7B on one 80GB A100 GPU, 4-bit QLoRA quantization was used. Furthermore, the context length was limited to 4096 tokes, which excluded prompts from one database in the BIRD training dataset, which required around 9000 tokens per prompt. Using 9000 tokens per prompt did not fit into memory during training with batch sizes larger than 1. 10 % of the training dataset was used for evaluation to observe how well the model learns, and the rest was used to fine-tune the model.

The SMoE gating layer contains few parameters compared to other layers, but has a big impact on performance. Therefore, it was initially planned to not quantize the gating layer, and use full fine-tuning on the gating layer but QLoRA on the other layers. However, the existing fine-tuning frameworks do not support mixing QLoRA and other fine-tuning methods in the same model. Therefore, QLoRA was used for all layers in the model, including the gating layer.

The final hyperparameters used are presented in Table 4.2. The QLoRA hyper-parameters were selected as recommended[66]. The batch size was selected to be as large as possible without memory overflow, a smaller and a larger learning rate was tested, but this was found best. The other parameters are standard, having an 8-bit optimizer instead of a 16-bit optimizer saves VRAM, but does not have a significant impact on quality [97].

Table 4.2: The hyperparameters used when fine-tuning Mixtral.

Hyperparameter	**Value**
Learning Rate	0.0002 (8-bit AdamW)
Learning Rate Scheduler	Cosine
Batch Size	2
Gradient Accumulation Steps	2
Lora R	64
Lora α	32
Lora Dropout	0.05
Epochs	2
Weight Decay	0
Warmup Steps	10

4.5.4 Evaluation

The models will be evaluated against baselines. These include the existing models evaluated on BIRD presented in Chapter 3, as well as a pre-trained baseline for Mixtral, which is used to show how much the fine-tuning process increases performance.

All three models, Mixtral FT, Mixtral baseline and Claude Opus were evaluated on the BIRD evaluation dataset, using the respective formats explained in Section 4.3. the FT Mixtral was evaluated both without Beam Search and using Beam Search with a width of 10 beams. The generation was never restricted to only generating valid SQL. Furthermore, the generation of the models was set to be deterministic, such that asking the same question always generates the same SQL query. However, although it is possible to select the variety of results generated by Claude Opus, there was no option to make it completely deterministic.

4.6 Statistical Methods

To evaluate the performance of the models, some statistical tools are used. The FT and pre-trained models are compared against each other using paired testing, and the scores reported for each individual test are the EX and VES scores, both reported with $95\,\%$ **C**onfidence **I**ntervals (CIs).

4.6.1 Confidence Intervals

To show the reliability of each estimate, CIs are used. The CIs are computed using a Z-score of 1.96, corresponding to a $95\,\%$ CI. The formula for computing CIs is shown in Equation 4.1, where $\bar{x}$ is the sample mean, s is the sample standard deviation and n is the number of samples.

$$\mathrm{CI} = \bar{x} \pm Z\frac{s}{\sqrt{n}} \tag{4.1}$$

4.6.2 Paired testing

To perform paired testing, statistical significance testing is used with a null hypothesis $H_0 : \bar{x}_1 = \bar{x}_2$ and an alternative hypothesis $H_a : \bar{x}_1 \neq \bar{x}_2$, where $\bar{x}_1$ is the mean result using one method and $\bar{x}_2$ the mean result of another method.

Some commonly used statistical tests are the Welch T-test, Student's T-test, and the Z-test. The Z-test assumes a normal distribution with known variance, while the T-test does not assume known variance. By the Central Limit Theorem (CLT) [98], the mean of a sum of Independent and Identically Distributed (IID) random variables converges to a normal distribution as the number of samples approaches infinity. When having over 30 samples, the distribution of the mean is considered to be a good approximation of the normal distribution, and the Z-test is often preferred [99]. Therefore, the Z-test is often used for large samples instead of the T-test. On the other hand, the T-test is more rejecting and therefore a slightly harder test to pass. The Welch T-test was chosen, which compared to the Students T-test is found to give more accurate predictions when the assumption of equal variance is not met [100]. The formula for computing the p-value, the probability of the observed results given that H_0 is correct, is shown in Equation 4.2 and Equation 4.3, where $t(\cdot)$ is the t-distribution and df the degrees of freedom. s is the sample standard deviation, $\hat{x}$ is the sample mean and n the number of samples.

$$T = \frac{\bar{x}_1 - \bar{x}_2}{\sqrt{\frac{s_1^2}{n_1} + \frac{s_2^2}{n_2}}} \tag{4.2}$$

$$p = t(df, T) \tag{4.3}$$

$$df = \frac{\left(s_1^2/n_1 + s_2^2/n_2\right)^2}{\frac{\left(s_1^2/n_1\right)^2}{n_1 - 1} + \frac{\left(s_2^2/n_2\right)^2}{n_2 - 1}} \tag{4.4}$$

It was decided to use a significance level of 0.05, hence any p-value below or equal to 0.05 rejects the null hypothesis H_0, and any p-value above 0.05 does not reject H_0. In other words, the hypothesis H_0 is rejected when the chance of the observed results are less than 5% under the assumption that H_0 is true.

Chapter 5

Results

In Table 5.1, the results of Mixtral and Claude Opus is shown. Mixtral FT with Beam Search and Claude Opus has a statistically insignificant difference in terms of EX score with a p-value of 0.8005. Furthermore, their VES is also similar.

Table 5.1: Results of Claude Opus and Mixtral, and the work presented in Chapter 3. CIs are reported with ± for the evaluated models.

Model	EX ↑	VES ↑
DIN-SQL+GPT-4 [7]	50.72	55.90
DAIL-SQL+GPT-4 [5]	54.76	57.41
MAC-SQL+GPT-4 [4]	57.56	59.59
Mixtral FT + Beam Search	50.52 ± 2.50	54.53 ± 5.85
Mixtral FT	46.61 ± 2.50	50.57 ± 5.77
Mixtral Baseline	33.57 ± 2.36	35.27 ± 3.38
Claude Opus	50.98 ± 2.50	55.79 ± 5.85

When comparing the EX grouped by difficulty as shown in Table 5.2, both models are also similar in performance. Although Claude Opus has slightly higher overall and grouped scores than Mixtral on all but the Challenging difficulty, the differences are insignificantly different, and it cannot be concluded that one model can be preferred over the other.

Table 5.2: EX presented for Mixtral FT with Beam Search and Claude Opus, grouped by difficulty of the questions.

Difficulty	Mixtral FT + Beam Search	Claude Opus
Simple	57.95 ± 3.18	58.81 ± 3.17
Moderate	41.51 ± 4.48	42.15 ± 4.49
Challenging	31.94 ± 7.62	29.17 ± 7.42

5.1 Baseline

Both Mixtral FT with Beam Search and Claude Opus perform similar to DIN-SQL [7], but are worse than DAIL-SQL [5] and MAC-SQL [4].

Comparing Mixtral FT and Mixtral FT with Beam Search against the Mixtral baseline shows that the fine-tuning process improved the model performance by a lot, making it competitive with methods based on GPT-4 and Claude Opus despite its much smaller size. Compared to the current best models, both Mixtral and Claude Opus are around 7 percent points behind the SOTA in terms of EX, and 5 percent points behind in terms of VES.

5.2 Error Analysis

The proportions of incorrectly classified data points, categorized based on what kind of error caused the incorrect classification is shown in Figure 5.1. Over 20 % of the incorrectly classified data had errors in the gold SQL. Outside of the incorrectly classified samples, Mixtral FT with Beam Search struggled the most with misunderstanding the utterance, evidence, or DDL schema, causing incorrect results on 239 data points, compared to Claude Opus which only had these misunderstands 117 times. Furthermore, Mixtral was worse than Claude Opus at generating executable SQL, where 68 queries could not be executed due to errors in the SQL, compared to Claude Opus with only 6 times, where 4 of those were due to exceeding the set 30 second per query time limit. Mixtral did not generate any queries which exceeded the time limit. Claude Opus often generated semantically correct queries, but with small differences in the returned data compared to the gold query. For example, Claude Opus always rounds fractions to two decimal places, while the gold queries do not. This is not classified as equal by the evaluation script. However, I consider it equivalent in the error analysis as the model has correctly answered the utterance, which does not specify if the result should be rounded or not.

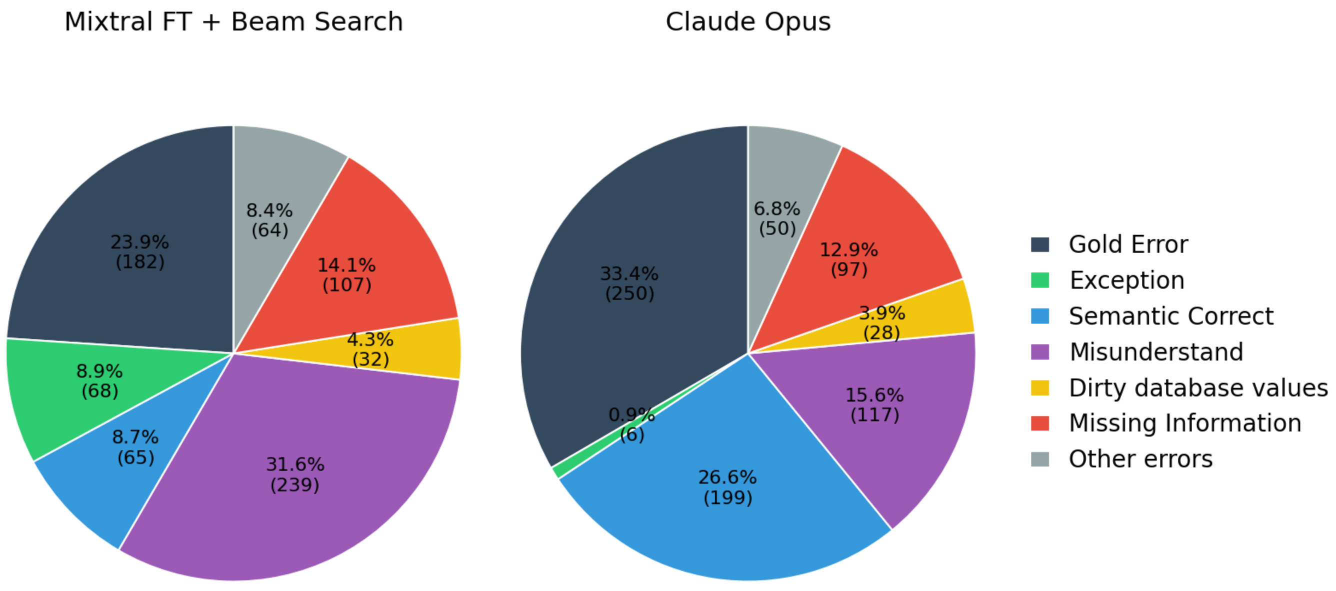

Figure 5.1: The incorrect samples for Mixtral FT + Beam Search and Claude Opus, sorted into categories based on what caused the mismatch between gold query and predicted query. In total, In total, there are 760 incorrect datapoints for Mixtral and 752 incorrect datapoints for Claude Opus.

With all Semantic Correct queries considered as correct, and removing all Gold Errors and Missing Information from the database, the EX increases by 17 % points and 32 % points for Mixtral and Claude Opus respectively. The Missing Information data points are removed as these utterances cannot be answered by the model, and must therefore be changed in the dataset or by giving the model more information such as database values. However, to find out how good the model is compared against the best possible performance of the model, data points in this category are not considered. The adjusted EX are shown in Table 5.3. Furthermore, comparing the performance of Mixtral and Claude Opus with the newly obtained EX, Claude Opus significantly outperforms Mixtral, with a p-value of $4.30 \cdot 10^{-23}$, concluding that the Claude Opus model is the better of the two models.

Table 5.3: EX of Mixtral FT with Beam Search and Claude Opus after removing incorrect gold queries and Missing Information datapoints from the evaluation data, and considering Semantically correct queries as correct.

Model	EX $\uparrow$
Mixtral FT + Beam Search	67.47 ± 2.60
Claude Opus	82.80 ± 2.15

Chapter 6

Discussion

In this section, the results of the Mixtral and Claude Opus will be discussed. The dataset will also be discussed in view of the error analysis. Furthermore, ethical and environmental impact is discussed.

6.1 Fine-Tuned Mixtral with Beam Search

Using the BIRD evaluation script, both Mixtral FT with Beam Search and Claude Opus are similar in performance, although the error analysis showed that Claude Opus is significantly better when the data is clean. One reason why Mixtral performs worse with clean data compared to Claude Opus is that if the training and evaluation data is similar, both containing the same types of gold errors, then the model will learn to mimic the common errors. The evaluation script will consider these errors in both gold and generated queries as being correct, classifying data points that are incorrect as correct. This could explain the lower gold error rate for Mixtral in Figure 5.1.

Perhaps fine-tuning the model on other higher quality Text-to-SQL data than the BIRD training dataset would have improved the model's Text-to-SQL ability. This is supported in [95], who show that the quality of data is very important when training LLMs.

Mixtral also showed a lack of understanding of natural language, having many misunderstandings. It is unclear if this is an effect of the small model size, or the fine-tuning process, or a combination of the two.

6.2 Claude Opus

Claude Opus significantly outperformed Mixtral when looking at the error analysis results. However, there are some potential issues with Claude Opus.

Firstly, the BIRD evaluation dataset was available before the training cutoff of Claude Opus. Therefore, there is a chance that the model could suffer from data contamination. In other programming-related datasets, it has been shown that LLMs perform significantly better on tasks from popular Github repositories that existed before the training cutoff [101], giving strong evidence of data contamination in the tested GPT-4 and Llama models. Furthermore, an example prediction by Claude which may suggest that the model has seen the data before is shown in Table 6.1, where the generated query correctly predicts that 'north' should be lowercase, without any information suggesting that. It could also be a case of the model having strong priors, where "north" is more common in its training dataset.

Table 6.1: Example 93 from the evaluation dataset. Claude Opus guess that north should be lowercase, which could be a sign of data contamination.

Utterance	How many male customers who are living in North Bohemia have average salary greater than 8000?
Evidence	Male means that gender = 'M'; A3 refers to region; A11 pertains to average salary.
Predicted query	`SELECT COUNT(*) AS count FROM client c JOIN district d ON c.district_id = d.district_id WHERE c.gender = 'M' AND d.A3 = 'north Bohemia' AND d.A11 > 8000;`

There were cases where the model was able to apply knowledge in the database area in its attempt to answer the questions. This was most evident in the Formula 1 database, which contains data mainly regarding Formula 1 seasons and races throughout the entire history of Formula 1. For example, Table 6.2 shows that the model has an understanding of Formula 1 races naming conventions, although it is not explicitly mentioned in the question.[*] Mixtral FT did not show this type of behaviour, and always copied the relevant information from the utterance as is.

Table 6.2: Example 922 from the evaluation dataset. The predicted query by Claude Opus shows knowledge in the Formula 1 domain by inferring that the race on Abu Dhabi Circuit is called Abu Dhabi Grand Prix. The predicted query is nevertheless incorrect, as it matches the race name against the circuit name.

Utterance	What time did the the 2010's Formula_1 race took place on the Abu Dhabi Circuit?
Evidence	-
Gold query	`SELECT T2.date, T2.time FROM circuits AS T1 INNER JOIN races AS T2 ON T2.circuitID = T1.circuitId WHERE T2.year = 2010 AND T2.name = 'Abu Dhabi Grand Prix'`
Predicted query	`SELECT time FROM races WHERE year = 2010 AND circuitId = ( SELECT circuitId FROM circuits WHERE name = 'Abu Dhabi Grand Prix' );`

6.3 The Error Analysis

Around 40 hours was spent studying the evaluation data classified as incorrect by the evaluation of Mixtral FT with Beam Search and Claude Opus. In some cases, it was difficult to separate the categories, for example, it can sometimes

[*]The utterance, evidence and gold query is copied directly from the evaluation dataset, including the grammatical error.

be hard to decide if the error comes from misinterpreting the question, or if the question was understood correctly but the mistakes stem other issues such as not being able to properly express the question in SQL.

Figuring out if gold queries are correct or not can be both difficult and time consuming, as it requires a good understanding of the database, as well as the utterance, evidence and gold query. Hence, there is a chance that some data points are incorrectly classified as gold errors, and that some are missed.

The EX from the error analysis only looks at data points incorrectly classified by the evaluation script. However, this excludes some incorrect gold queries where the model managed to predict the same, incorrect result. When considering the data points considered correct by the evaluation script for one model, but considered as gold errors by the error analysis for the other model, the score of Mixtral decrease by $0.3\,\%$ points. This is because a proportion of the classifications considered correct by the evaluation script will instead be considered as incorrect.

6.4 The Dataset

As found in the error analysis, the dataset is not perfect. A study has been done on some data points from some of the databases in the BIRD evaluation dataset [102], and found a 5–40 % gold query error rate depending on the database. Dividing the incorrect gold queries from the error analysis by database as shown in Table 6.3, gives a 7–26 % gold query error rate depending on the database.

Out of the 937 data points covered by the error analysis, 265 had an incorrect gold query. Hence, making up $28\,\%$ of the considered queries and $17\,\%$ of all data. Although not all queries was seen by the error analysis, all samples not seen by the study had matching execution across both gold, Mixtral and Claude Opus, making these less likely to have errors, as both models predicted the same answer as the gold answer, but often using different queries. Hence, the 17 % gold error rate can be considered as close to the true error rate of the dataset.

Among the incorrect gold queries, there were some recurring types of errors. Some common errors are shown in Table 6.4. Example 1 shows incorrect usage of limiting to one result when the utterance asks for entries which contains MAX or MIN in a column. Using `LIMIT 1` in this scenario is only true under the incorrect assumption that there is only one such entry. Another common

Table 6.3: Incorrect Gold queries divided by database.

Database	Incorrect Gold
California Schools	8.99 %
Financial	24.53 %
Toxicology	20.0 %
Card Games	14.66 %
Codebase Community	17.74 %
Superhero	9.30 %
Formula 1	24.14 %
European Football 2	16.28 %
Thrombosis Prediction	26.38 %
Student Club	7.59 %
Debit Card Specializing	17.19 %
Total	**17.28 %**

recurring issue is that filtering or ordering based on dates or times is sometimes done incorrectly. Furthermore, sometimes the `DISTINCT` keyword is missing when asked to list ids, causing the same entry to be listed multiple times although the utterance explicitly asks for each entry only once. Another mistake is that strings which had words ending with `and` were converted into uppercase, making the queries incorrect. This could be the effect of a post processing step where the SQL keywords are highlighted. Furthermore, a common error in some databases were that the gold query joined foreign keys in an incorrect way, for example joining two tables based on a common foreign key, instead of using a relational table linking the two tables together. This was most common in the `toxicology` database. The relational structure of the database is shown in Figure 6.1. In this database, `BOND` and `ATOM` are linked together via the relational table `CONNECTED`, specifying which two atoms are linked via which bond. However, in gold queries such as the one presented in Table 6.5, the gold query incorrectly joins `BOND` and `ATOM` via `molecule_id` instead of via the `CONNECTED` relation, creating a row for each atom and bond combination, without restricting it to those which are bonded together within the molecule, making the gold query list all elements which exist in any molecule that has a double type bond. Similar types of database misunderstandings can be seen in some of the other evaluation

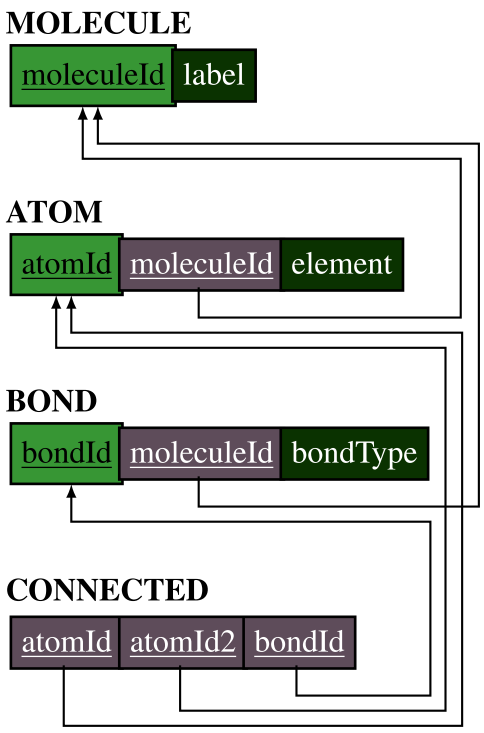

Figure 6.1: Database schema for the toxicology database. The arrows point from foreign keys to primary keys. The connected table specifies what bond holds what atoms together.

databases as well.

When using the training data in the prompt designing process, it was found that there were some errors in the training data as well. 20 data points was studied from two databases, and out of these there were 6 incorrect gold queries, indicating that the training data is of similar quality as the evaluation dataset.

The language used in the utterances and evidences is sometimes lacking in clarity and structure. [3] created the dataset to be noisy and have such errors, as these are issues which are expected to be present in real world scenarios when using a Text-to-SQL tool. However, such language could negatively impact the FT process, which could be one of the reasons why Mixtral has a large proportion of misunderstandings in the error analysis.

6.4.1 Evaluation

The evaluation script available by BIRD misses some correct classifications as evident by Section 5.2. However, there is no simple solution to automating evaluation such that the script can detect all semantically correct samples without also misclassifying incorrect samples as correct without manual evaluation. For example, if the gold query returns string `Y` and the model returns `YES`, then the result should be considered semantically correct. Another example missed by the evaluation script is that answering some Boolean question correctly can be done without answering the utterance, by generating a query that does not answer the utterance but returns true or false depending on a different question that happens to have the same answer as the correct gold query. In this case, the query is semantically incorrect although it gives the right answer. These issues could be problematic and are difficult to solve. However, perhaps the simplest solution is to avoid utterances with Boolean answers, and instead ask that a certain row be returned, making it a lot more difficult to accidentally return the correct answer without answering the utterance.

6.5 Ethical Considerations

There are some dangers in providing Text-to-SQL as a service. For example, undetected incorrectly generated output could have devastating effects if the resulting query is used to make important business decisions. This issue creates a problem regarding responsibility: Should the creators of the Text-to-SQL service be held accountable, or should the person generating the query be liable for not detecting the error? To avoid confusion, it should be clearly stated that the generated SQL can be incorrect and whose responsibility it is to ensure that the generated queries are correct.

With an accuracy of 82 %, Claude Opus could be a useful tool for generating SQL. However, it will make mistakes, often ones which are not easily detectable by only studying the output when executing the SQL. Hence, it must be used with care and the users of such a system should have access to the SQL queries to be able to validate that the generated query is correct.

The BIRD dataset uses open and accessible databases, and the databases do not contain any confidential information. All information in the databases in BIRD is public and accessible [3].

6.6 Environmental Impacts

The sustainability aspect should also be considered. Training LLMs can use a lot of electricity. The training of GPT-3 emitted around 500 tonnes of CO_2eq and consumed over 1 GWh of electricity [58]. It is therefore important to keep in mind the environmental impact of training and testing LLMs. However, FT and inference consume a lot less electricity compared to the pre-training.

The A100 GPU used in this study consumes 300W at maximum usage, and the FT process of Mixtral took roughly 3 hours. The evaluation process took roughly 5 hours, but with lower energy consumption as the GPU was not fully utilized. The evaluation process was run three times, once before FT and once with and without beam search. Hence, a rough estimate for the electricity consumption is $18\text{h} \times 300\text{W} = 5.4\text{kWh}$. Converting the consumed energy into CO_2eq accurately is nontrivial as there does not exist any public information regarding the Power Usage Effectiveness (PUE) of the data center, nor the percentage of renewable electricity used in the data center. As an estimate, the range of carbon intensity of the energy grid and PUE used by the studied pre-trained models in [58] is used. The carbon intensity of the energy grid is in the range of 57-429gCO_2eq/kWh and the data center PUE is in the range 1.08 to 1.2. Using these numbers, the estimated carbon emissions from the Mixtral FT and evaluation process are in the range of 332g-2 780g of CO_2eq.

Table 6.4: The Question, Evidence, and Gold query for two examples in the evaluation dataset are shown, together with an explanation why the Gold query is incorrect.

	Example 1	Example 2
Utterance	List the players' api id who had the highest above average overall ratings in 2010.	List the driver's ID of the top five driver, by descending order, the fastest time during the first lap of the race.
Evidence	highest above average overall ratings refers to MAX(overall_rating); in 2010 refers to substr(date,1,4) = '2010';	fastest time refers to Min(time);
Gold query	`SELECT player_api_id FROM Player_Attributes WHERE SUBSTR('date', 1, 4) = '2010' ORDER BY overall_rating DESC LIMIT 1`	`SELECT driverId FROM lapTimes WHERE lap = 1 ORDER BY time LIMIT 5`
Explanation	Multiple players has the same highest rating, should list all of them instead of using `LIMIT 1`.	`time` is a string of the form `minutes:ss.ms`, where minutes are written without a leading 0, i.e. '2' or '10'. When sorting ascending, minute 10-19 is ordered before minute 1 and therefore the gold query does not return the driverIds of the fastest drivers.

Table 6.5: Example 207 from the evaluation dataset. The gold query incorrectly joins atom and bond using molecule_id before joining with the connected table.

Utterance	What elements are in a double type bond?
Evidence	double type bond refers to bond_type = ' = '; element = 'cl' means Chlorine; element = 'c' means Carbon; element = 'h' means Hydrogen; element = 'o' means Oxygen, element = 's' means Sulfur; element = 'n' means Nitrogen, element = 'p' means Phosphorus, element = 'na' means Sodium, element = 'br' means Bromine, element = 'f' means Fluorine; element = 'i' means Iodine; element = 'sn' means Tin; element = 'pb' means Lead; element = 'te' means Tellurium; element = 'ca' means Calcium
Gold query	`SELECT DISTINCT T1.element FROM atom AS T1 INNER JOIN bond AS T2 ON T1.molecule_id = T2.molecule_id INNER JOIN connected AS T3 ON T1.atom_id = T3.atom_id WHERE T2.bond_type = '='`

Chapter 7

Conclusions and Future work

In conclusion, Claude Opus significantly outperforms Mixtral, with accuracies of 82 % and 67 %, respectively, when removing incorrect data points and adding semantically correct data points missed by the evaluation script. The BIRD evaluation dataset contains roughly 17 % gold SQL errors, and most likely a similar proportion of errors in the training data, which impacted the fine-tuning process of Mixtral. A strong understanding of language is important, and Mixtral lacks this understanding after the fine-tuning process, and Claude Opus does not. This becomes especially important when the utterances contain complex logic and require a deep understanding of the language. With a 82 % accuracy, using Claude Opus for SQL generation could be a useful tool for programmers in creating SQL queries.

The main contribution of the thesis highlight that the data quality of the entire BIRD evaluation dataset contains 17 % gold SQL errors and data points that cannot be answered without ambiguity due to missing or incorrect information in the utterance or evidence. In total, 937 out of 1537 data points were covered by the Error Analysis, compared to previous work which has studied 186 data points [102].

7.1 Future Work

In this thesis, both fine-tuning Mixtral on BIRD to improve its Text-to-SQL performance and using pre-trained Claude Opus have been evaluated on the BIRD evaluation dataset. In future work, it would be beneficial to fine-tune models on other Text-to-SQL datasets in order to explore fine-tuning further, as high quality data is of importance in the fine-tuning process. Furthermore, using more advanced prompting techniques for Claude Opus could increase performance, for example by providing example column values to the model and descriptions of the meaning of each column. Furthermore, Few-Shot prompting techniques could also be tried, although they are not as cost-effective. It would also be interesting to test Claude Opus on the Spider dataset.

References

[1] G. G. Hendrix, E. D. Sacerdoti, D. Sagalowicz, and J. Slocum, "Developing a natural language interface to complex data," *ACM Transactions on Database Systems (TODS)*, vol. 3, no. 2, pp. 105–147, 1978. [Pages 1, 7, and 8.]

[2] E. F. Codd, "A relational model of data for large shared data banks," *Communications of the ACM*, vol. 13, no. 6, pp. 377–387, 1970. [Pages 1 and 5.]

[3] J. Li, B. Hui, G. Qu, B. Li, J. Yang, B. Li, B. Wang, B. Qin, R. Cao, R. Geng *et al.*, "Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls," *arXiv preprint arXiv:2305.03111*, 2023. [Pages 2, 28, 29, 30, 60, and 61.]

[4] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, Q.-W. Zhang, Z. Yan, and Z. Li, "Mac-sql: Multi-agent collaboration for text-to-sql," *arXiv preprint arXiv:2312.11242*, 2023. [Pages 2, 12, 25, 34, 35, 36, 37, 51, and 52.]

[5] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023. [Pages 2, 25, 34, 35, 36, 37, 51, and 52.]

[6] J. Achiam, S. Adler, S. Agarwal, L. Ahmad, I. Akkaya, F. L. Aleman, D. Almeida, J. Altenschmidt, S. Altman, S. Anadkat *et al.*, "Gpt-4 technical report," *arXiv preprint arXiv:2303.08774*, 2023. [Pages 2, 13, 18, and 22.]

[7] M. Pourreza and D. Rafiei, "Din-sql: Decomposed in-context learning of text-to-sql with self-correction," *arXiv preprint arXiv:2304.11015*, 2023. [Pages 2, 12, 25, 33, 36, 37, 51, and 52.]

[8] T. Liu and B. K. H. Low, "Goat: Fine-tuned llama outperforms gpt-4 on arithmetic tasks," *arXiv preprint arXiv:2305.14201*, 2023. [Page 2.]

[9] J. Shin, C. Tang, T. Mohati, M. Nayebi, S. Wang, and H. Hemmati, "Prompt engineering or fine tuning: An empirical assessment of large language models in automated software engineering tasks," *arXiv preprint arXiv:2310.10508*, 2023. [Page 2.]

[10] A. Pavlo and M. Aslett, "What's really new with newsql?" *ACM Sigmod Record*, vol. 45, no. 2, pp. 45–55, 2016. [Page 5.]

[11] H. Sato, "A data model, knowledge base, and natural language processing for sharing a large statistical database," in *Statistical and Scientific Database Management: Fourth International Working Conference SSDBM Rome, Italy, June 21–23, 1988 Proceedings 4*. Springer, 1989, pp. 207–225. [Pages 7 and 8.]

[12] A.-M. Popescu, O. Etzioni, and H. Kautz, "Towards a theory of natural language interfaces to databases," in *Proceedings of the 8th international conference on Intelligent user interfaces*, 2003, pp. 149–157. [Page 7.]

[13] W.-C. Kwan, X. Zeng, Y. Wang, Y. Sun, L. Li, L. Shang, Q. Liu, and K.-F. Wong, "M4le: A multi-ability multi-range multi-task multi-domain long-context evaluation benchmark for large language models," *arXiv preprint arXiv:2310.19240*, 2023. [Page 8.]

[14] X. Xu, C. Liu, and D. Song, "Sqlnet: Generating structured queries from natural language without reinforcement learning," *arXiv preprint arXiv:1711.04436*, 2017. [Pages 8 and 11.]

[15] V. Zhong, C. Xiong, and R. Socher, "Seq2sql: Generating structured queries from natural language using reinforcement learning," *arXiv preprint arXiv:1709.00103*, 2017. [Pages 8, 11, 29, and 30.]

[16] T. Yu, M. Yasunaga, K. Yang, R. Zhang, D. Wang, Z. Li, and D. Radev, "Syntaxsqlnet: Syntax tree networks for complex and cross-domaintext-to-sql task," *arXiv preprint arXiv:1810.05237*, 2018. [Pages 8 and 11.]

[17] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL

parsers," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Online: Association for Computational Linguistics, Jul. 2020, pp. 7567–7578. [Pages 9, 10, and 11.]

[18] M.-C. De Marneffe and J. Nivre, "Dependency grammar," *Annual Review of Linguistics*, vol. 5, pp. 197–218, 2019. [Page 10.]

[19] B. Bogin, M. Gardner, and J. Berant, "Representing schema structure with graph neural networks for text-to-sql parsing," *arXiv preprint arXiv:1905.06241*, 2019. [Page 10.]

[20] P. Shaw, J. Uszkoreit, and A. Vaswani, "Self-attention with relative position representations," *arXiv preprint arXiv:1803.02155*, 2018. [Page 10.]

[21] N. M. Ndongala, "Light rat-sql: A rat-sql with more abstraction and less embedding of pre-existing relations," *Texila Int. J. Acad. Res*, vol. 10, no. 2, pp. 1–11, 2023. [Page 10.]

[22] J. Qi, J. Tang, Z. He, X. Wan, Y. Cheng, C. Zhou, X. Wang, Q. Zhang, and Z. Lin, "Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql," *arXiv preprint arXiv:2205.06983*, 2022. [Page 10.]

[23] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li, "Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing," *arXiv preprint arXiv:2301.07507*, 2023. [Pages 10, 12, 34, and 37.]

[24] T. Scholak, N. Schucher, and D. Bahdanau, "Picard: Parsing incrementally for constrained auto-regressive decoding from language models," *arXiv preprint arXiv:2109.05093*, 2021. [Pages 11 and 12.]

[25] K. Lin, B. Bogin, M. Neumann, J. Berant, and M. Gardner, "Grammar-based neural text-to-sql generation," *arXiv preprint arXiv:1905.13326*, 2019. [Page 11.]

[26] L. Dou, Y. Gao, M. Pan, D. Wang, W. Che, D. Zhan, and J.-G. Lou, "Unisar: A unified structure-aware autoregressive language model for text-to-sql," *arXiv preprint arXiv:2203.07781*, 2022. [Pages 11 and 12.]

[27] Y. Xie, K. Kawaguchi, Y. Zhao, X. Zhao, M.-Y. Kan, J. He, and Q. Xie, “Self-evaluation guided beam search for reasoning,” in *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. [Page 12.]

[28] Y. Ge, W. Hua, J. Ji, J. Tan, S. Xu, and Y. Zhang, “Openagi: When llm meets domain experts,” *arXiv preprint arXiv:2304.04370*, 2023. [Page 12.]

[29] N. De Cao, G. Izacard, S. Riedel, and F. Petroni, “Autoregressive entity retrieval,” *arXiv preprint arXiv:2010.00904*, 2020. [Page 12.]

[30] R. Pryzant, D. Iter, J. Li, Y. T. Lee, C. Zhu, and M. Zeng, “Automatic prompt optimization with” gradient descent” and beam search,” *arXiv preprint arXiv:2305.03495*, 2023. [Page 12.]

[31] K. Xu, Y. Wang, Y. Wang, Z. Wen, and Y. Dong, “Sead: End-to-end text-to-sql generation with schema-aware denoising,” *arXiv preprint arXiv:2105.07911*, 2021. [Page 12.]

[32] J. Austin, A. Odena, M. Nye, M. Bosma, H. Michalewski, D. Dohan, E. Jiang, C. Cai, M. Terry, Q. Le *et al.*, “Program synthesis with large language models,” *arXiv preprint arXiv:2108.07732*, 2021. [Page 12.]

[33] A. Ni, S. Iyer, D. Radev, V. Stoyanov, W.-t. Yih, S. Wang, and X. V. Lin, “Lever: Learning to verify language-to-code generation with execution,” in *International Conference on Machine Learning*. PMLR, 2023, pp. 26 106–26 128. [Pages 12 and 25.]

[34] C. Guo, Z. Tian, J. Tang, S. Li, Z. Wen, K. Wang, and T. Wang, “Retrieval-augmented gpt-3.5-based text-to-sql framework with sample-aware prompting and dynamic revision chain,” in *International Conference on Neural Information Processing*. Springer, 2023, pp. 341–356. [Pages 12, 25, 35, and 37.]

[35] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou, “Self-consistency improves chain of thought reasoning in language models,” *arXiv preprint arXiv:2203.11171*, 2022. [Page 13.]

[36] J. Wei, X. Wang, D. Schuurmans, M. Bosma, F. Xia, E. Chi, Q. V. Le, D. Zhou *et al.*, “Chain-of-thought prompting elicits reasoning in

large language models," *Advances in Neural Information Processing Systems*, vol. 35, pp. 24 824–24 837, 2022. [Page 13.]

[37] R. Sun, S. O. Arik, H. Nakhost, H. Dai, R. Sinha, P. Yin, and T. Pfister, "Sql-palm: Improved large language modeladaptation for text-to-sql," *arXiv preprint arXiv:2306.00739*, 2023. [Pages 13, 25, 35, 36, and 37.]

[38] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, J. Lin, D. Lou *et al.*, "C3: Zero-shot text-to-sql with chatgpt," *arXiv preprint arXiv:2307.07306*, 2023. [Pages 13, 33, 36, and 37.]

[39] G. Team, R. Anil, S. Borgeaud, Y. Wu, J.-B. Alayrac, J. Yu, R. Soricut, J. Schalkwyk, A. M. Dai, A. Hauth *et al.*, "Gemini: a family of highly capable multimodal models," *arXiv preprint arXiv:2312.11805*, 2023. [Page 13.]

[40] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020. [Pages 13 and 15.]

[41] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, "Improving language understanding by generative pre-training," 2018. [Pages 13, 15, and 18.]

[42] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever *et al.*, "Language models are unsupervised multitask learners," *OpenAI blog*, vol. 1, no. 8, p. 9, 2019. [Pages 13 and 15.]

[43] M. Schuster and K. Nakajima, "Japanese and korean voice search," in *2012 IEEE international conference on acoustics, speech and signal processing (ICASSP)*. IEEE, 2012, pp. 5149–5152. [Page 14.]

[44] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016. [Page 14.]

[45] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018. [Page 15.]

[46] P. Gage, "A new algorithm for data compression," *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994. [Page 15.]

[47] Y. Shibata, T. Kida, S. Fukamachi, M. Takeda, A. Shinohara, T. Shinohara, and S. Arikawa, "Byte pair encoding: A text compression scheme that accelerates pattern matching," 1999. [Page 15.]

[48] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer, "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension," *arXiv preprint arXiv:1910.13461*, 2019. [Page 15.]

[49] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," *Advances in neural information processing systems*, vol. 26, 2013. [Page 15.]

[50] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, 2014, pp. 1532–1543. [Page 15.]

[51] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," *Advances in neural information processing systems*, vol. 30, 2017. [Pages 16, 17, 18, 20, 21, and 41.]

[52] A. Haviv, O. Ram, O. Press, P. Izsak, and O. Levy, "Transformer language models without positional encodings still learn positional information," *arXiv preprint arXiv:2203.16634*, 2022. [Page 16.]

[53] J. Zheng, S. Ramasinghe, and S. Lucey, "Rethinking positional encoding," *arXiv preprint arXiv:2107.02561*, 2021. [Page 16.]

[54] J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, "Roformer: Enhanced transformer with rotary position embedding," *Neurocomputing*, vol. 568, p. 127063, 2024. [Pages 16 and 41.]

[55] P.-C. Chen, H. Tsai, S. Bhojanapalli, H. W. Chung, Y.-W. Chang, and C.-S. Ferng, "A simple and effective positional encoding for transformers," *arXiv preprint arXiv:2104.08698*, 2021. [Page 16.]

[56] D. Bahdanau, K. Cho, and Y. Bengio, “Neural machine translation by jointly learning to align and translate,” *arXiv preprint arXiv:1409.0473*, 2014. [Page 16.]

[57] K. Clark, U. Khandelwal, O. Levy, and C. D. Manning, “What does bert look at? an analysis of bert’s attention,” *arXiv preprint arXiv:1906.04341*, 2019. [Page 17.]

[58] A. S. Luccioni, S. Viguier, and A.-L. Ligozat, “Estimating the carbon footprint of bloom, a 176b parameter language model,” *Journal of Machine Learning Research*, vol. 24, no. 253, pp. 1–15, 2023. [Pages 22 and 62.]

[59] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, and Q. He, “A comprehensive survey on transfer learning,” *Proceedings of the IEEE*, vol. 109, no. 1, pp. 43–76, 2020. [Page 22.]

[60] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, T. Remez, J. Rapin *et al.*, “Code llama: Open foundation models for code,” *arXiv preprint arXiv:2308.12950*, 2023. [Page 22.]

[61] E. J. Hu, Y. Shen, P. Wallis, Z. Allen-Zhu, Y. Li, S. Wang, L. Wang, and W. Chen, “Lora: Low-rank adaptation of large language models,” *arXiv preprint arXiv:2106.09685*, 2021. [Pages 22, 23, and 24.]

[62] N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. De Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly, “Parameter-efficient transfer learning for nlp,” in *International Conference on Machine Learning*. PMLR, 2019, pp. 2790–2799. [Page 23.]

[63] A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. R. Bowman, “Glue: A multi-task benchmark and analysis platform for natural language understanding,” *arXiv preprint arXiv:1804.07461*, 2018. [Page 23.]

[64] Y. Wang, S. Mukherjee, X. Liu, J. Gao, A. H. Awadallah, and J. Gao, “Adamix: Mixture-of-adapter for parameter-efficient tuning of large language models,” *arXiv preprint arXiv:2205.12410*, vol. 1, no. 2, p. 4, 2022. [Page 23.]

[65] A. Aghajanyan, L. Zettlemoyer, and S. Gupta, “Intrinsic dimensionality explains the effectiveness of language model fine-tuning,” *arXiv preprint arXiv:2012.13255*, 2020. [Page 23.]

[66] T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, "Qlora: Efficient finetuning of quantized llms," *arXiv preprint arXiv:2305.14314*, 2023. [Pages 24, 41, and 46.]

[67] O. Rubin, J. Herzig, and J. Berant, "Learning to retrieve prompts for in-context learning," *arXiv preprint arXiv:2112.08633*, 2021. [Page 25.]

[68] Y. Lu, M. Bartolo, A. Moore, S. Riedel, and P. Stenetorp, "Fantastically ordered prompts and where to find them: Overcoming few-shot prompt order sensitivity," *arXiv preprint arXiv:2104.08786*, 2021. [Page 25.]

[69] W. Fedus, J. Dean, and B. Zoph, "A review of sparse expert models in deep learning," *arXiv preprint arXiv:2209.01667*, 2022. [Pages 25 and 26.]

[70] N. Shazeer, A. Mirhoseini, K. Maziarz, A. Davis, Q. Le, G. Hinton, and J. Dean, "Outrageously large neural networks: The sparsely-gated mixture-of-experts layer," *arXiv preprint arXiv:1701.06538*, 2017. [Page 26.]

[71] B. Zoph, I. Bello, S. Kumar, N. Du, Y. Huang, J. Dean, N. Shazeer, and W. Fedus, "St-moe: Designing stable and transferable sparse expert models," *arXiv preprint arXiv:2202.08906*, 2022. [Page 26.]

[72] F. Xue, Z. Zheng, Y. Fu, J. Ni, Z. Zheng, W. Zhou, and Y. You, "Openmoe: An early effort on open mixture-of-experts language models," *arXiv preprint arXiv:2402.01739*, 2024. [Page 26.]

[73] A. Q. Jiang, A. Sablayrolles, A. Roux, A. Mensch, B. Savary, C. Bamford, D. S. Chaplot, D. d. l. Casas, E. B. Hanna, F. Bressand *et al.*, "Mixtral of experts," *arXiv preprint arXiv:2401.04088*, 2024. [Pages 26 and 42.]

[74] A. Eliseev and D. Mazur, "Fast inference of mixture-of-experts language models with offloading," *arXiv preprint arXiv:2312.17238*, 2023. [Page 26.]

[75] T. Yu, R. Zhang, K. Yang, M. Yasunaga, D. Wang, Z. Li, J. Ma, I. Li, Q. Yao, S. Roman *et al.*, "Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task," *arXiv preprint arXiv:1809.08887*, 2018. [Pages 27, 29, and 30.]

[76] A. P. Zijdenbos, B. M. Dawant, R. A. Margolin, and A. C. Palmer, “Morphometric analysis of white matter lesions in mr images: method and validation,” *IEEE transactions on medical imaging*, vol. 13, no. 4, pp. 716–724, 1994. [Page 27.]

[77] D. Chicco and G. Jurman, “The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation,” *BMC genomics*, vol. 21, no. 1, pp. 1–13, 2020. [Page 27.]

[78] W. Chen, H. Zha, Z. Chen, W. Xiong, H. Wang, and W. Wang, “Hybridqa: A dataset of multi-hop question answering over tabular and textual data,” *arXiv preprint arXiv:2004.07347*, 2020. [Page 29.]

[79] T. Yu, R. Zhang, M. Yasunaga, Y. C. Tan, X. V. Lin, S. Li, H. Er, I. Li, B. Pang, T. Chen *et al.*, “Sparc: Cross-domain semantic parsing in context,” *arXiv preprint arXiv:1906.02285*, 2019. [Page 29.]

[80] C.-H. Lee, O. Polozov, and M. Richardson, “Kaggledbqa: Realistic evaluation of text-to-sql parsers,” *arXiv preprint arXiv:2106.11455*, 2021. [Page 29.]

[81] T. Yu, R. Zhang, H. Y. Er, S. Li, E. Xue, B. Pang, X. V. Lin, Y. C. Tan, T. Shi, Z. Li *et al.*, “Cosql: A conversational text-to-sql challenge towards cross-domain natural language interfaces to databases,” *arXiv preprint arXiv:1909.05378*, 2019. [Page 29.]

[82] A. Elgohary, S. Hosseini, and A. H. Awadallah, “Speak to your parser: Interactive text-to-sql with natural language feedback,” *arXiv preprint arXiv:2005.02539*, 2020. [Page 29.]

[83] M. Hazoom, V. Malik, and B. Bogin, “Text-to-sql in the wild: A naturally-occurring dataset based on stack exchange data,” *arXiv preprint arXiv:2106.05006*, 2021. [Page 29.]

[84] Y. Zhang, X. Dong, S. Chang, T. Yu, P. Shi, and R. Zhang, “Did you ask a good question? a cross-domain question intention classification benchmark for text-to-sql,” *arXiv preprint arXiv:2010.12634*, 2020. [Page 29.]

[85] X. Pi, B. Wang, Y. Gao, J. Guo, Z. Li, and J.-G. Lou, “Towards robustness of text-to-sql models against natural and realistic adversarial table perturbation,” *arXiv preprint arXiv:2212.09994*, 2022. [Page 29.]

[86] A. Liu, X. Hu, L. Wen, and P. S. Yu, "A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability," *arXiv preprint arXiv:2303.13547*, 2023. [Pages 33 and 37.]

[87] Y. Gan, X. Chen, J. Xie, M. Purver, J. R. Woodward, J. Drake, and Q. Zhang, "Natural sql: Making sql easier to infer from natural language specifications," *arXiv preprint arXiv:2109.05153*, 2021. [Page 33.]

[88] R. Anil, A. M. Dai, O. Firat, M. Johnson, D. Lepikhin, A. Passos, S. Shakeri, E. Taropa, P. Bailey, Z. Chen *et al.*, "Palm 2 technical report," *arXiv preprint arXiv:2305.10403*, 2023. [Pages 35 and 36.]

[89] D. Hendrycks, C. Burns, S. Basart, A. Zou, M. Mazeika, D. Song, and J. Steinhardt, "Measuring massive multitask language understanding," *arXiv preprint arXiv:2009.03300*, 2020. [Page 40.]

[90] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021. [Page 40.]

[91] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar *et al.*, "Llama: Open and efficient foundation language models," *arXiv preprint arXiv:2302.13971*, 2023. [Page 41.]

[92] H. Touvron, L. Martin, K. Stone, P. Albert, A. Almahairi, Y. Babaei, N. Bashlykov, S. Batra, P. Bhargava, S. Bhosale *et al.*, "Llama 2: Open foundation and fine-tuned chat models," *arXiv preprint arXiv:2307.09288*, 2023. [Page 41.]

[93] N. Shazeer, "Glu variants improve transformer," *arXiv preprint arXiv:2002.05202*, 2020. [Page 41.]

[94] S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and A. Awadallah, "Orca: Progressive learning from complex explanation traces of gpt-4," *arXiv preprint arXiv:2306.02707*, 2023. [Page 41.]

[95] S. Gunasekar, Y. Zhang, J. Aneja, C. C. T. Mendes, A. Del Giorno, S. Gopi, M. Javaheripi, P. Kauffmann, G. de Rosa, O. Saarikivi *et al.*, "Textbooks are all you need," *arXiv preprint arXiv:2306.11644*, 2023. [Pages 41 and 55.]

[96] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier *et al.*, “Mistral 7b,” *arXiv preprint arXiv:2310.06825*, 2023. [Page 41.]

[97] T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, “8-bit optimizers via block-wise quantization,” *arXiv preprint arXiv:2110.02861*, 2021. [Page 46.]

[98] S. G. Kwak and J. H. Kim, “Central limit theorem: the cornerstone of modern statistics,” *Korean journal of anesthesiology*, vol. 70, no. 2, p. 144, 2017. [Page 48.]

[99] P. Mishra, U. Singh, C. M. Pandey, P. Mishra, and G. Pandey, “Application of student’s t-test, analysis of variance, and covariance,” *Annals of cardiac anaesthesia*, vol. 22, no. 4, pp. 407–411, 2019. [Page 48.]

[100] M. Delacre, D. Lakens, and C. Leys, “Why psychologists should by default use welch’s t-test instead of student’s t-test,” *International Review of Social Psychology*, vol. 30, no. 1, pp. 92–101, 2017. [Page 48.]

[101] M. Roberts, H. Thakur, C. Herlihy, C. White, and S. Dooley, “To the cutoff... and beyond? a longitudinal perspective on llm data contamination,” in *The Twelfth International Conference on Learning Representations*, 2023. [Page 56.]

[102] N. Wretblad, F. G. Riseby, R. Biswas, A. Ahmadi, and O. Holmström, “Understanding the effects of noise in text-to-sql: An examination of the bird-bench benchmark,” *arXiv preprint arXiv:2402.12243*, 2024. [Pages 58 and 65.]

Appendix A

Prompts used for LLMs

In this Appendix, the prompts for Claude Opus, Mixtral FT and Pre-Trained Mixtral is shown.

A.1 Claude Opus

```
Your objective is to construct an SQLite query from
the given question, schema and evidence. The goal
is to retrieve the requested information accurately
and efficiently, with minimal complexity. If
multiple entries satisfy the same maximum or
minimum condition, all matching entries should be
included in the result. The SQL query should be put
within an ´´´sql ´´´ block. The provided database
schema is automatically generated and correctly
reflect the database structure

Schema:
{schema}
Question: {question}
Evidence: {evidence}
```

```
Analyze the question and rephrase what is asked in
your own words. Then, divide the question into sub
queries if necessary, considering the constraints,
and generate the final SQL after thinking step by
step:
```

A.2 Fine-Tuned Mixtral

```
Objective: Construct a SQLite query based on the
given question, schema, and evidence. The goal is
to retrieve the requested information accurately
and efficiently, with minimal complexity.

Question: {question}
This is the question we need to translate into an
SQL statement. It defines the information we are
seeking from the database.

Schema Information:
{schema}
This section describes the database schema,
including table names, field names, data types, and
any constraints. This information is critical for
constructing accurate SQL queries that adhere to
the database structure.

Evidence Supporting the Question:
{evidence}
This evidence supports the question, providing
context or additional information that may influence
how the SQL query is structured. It might include
specific conditions, data points, or relationships
mentioned in the question that are crucial for
formulating the correct SQL statement.
```

A.3 Pre-Trained Mixtral

The pre-trained Mixtral uses two prompts, one which is used initially, and one which is used when the generated SQL is not executable, to attempt to fix the incorrectly generated SQL. The prompt used intially is:

```
Objective: Construct a SQLite query based on the
given question, schema, and evidence. The goal is
to retrieve the requested information accurately
and efficiently, with minimal complexity.

Question: {question}
This is the query we need to translate into a SQL
statement. It defines the information we are seeking
from the database.

Schema Information:
{schema}
This section describes the database schema,
including table names, field names, data types, and
any constraints. This information is critical for
constructing accurate SQL queries that adhere to
the database structure.

Evidence Supporting the Question:
{evidence}
This evidence supports the question, providing
context or additional information that may influence
how the SQL query is structured. It might include
specific conditions, data points, or relationships
mentioned in the question that are crucial for
formulating the correct SQL statement.

Instructions:
1. Direct Data Access: Prioritize accessing data
directly related to the question's requirements.
Assess whether each piece of the query, especially
table accesses, directly contributes to answering
the question.
```

```
2. Judicious Use of SQL Elements: Each component of
the SQL query (e.g., SELECT, FROM, WHERE, etc.)
should have a clear purpose and direct relevance to
the question. Assess the necessity of joins and
other complex structures based on their direct
contribution to obtaining the required information.
3. Efficiency and Precision: Aim for the most
efficient path to retrieve the needed data, which
often means minimizing complexity. Ensure the query
is as concise as possible while still being
complete and accurate.
4. Column Selection: Select only the columns that
are directly necessary to answer the question.
Avoid including additional columns that do not
contribute to the specific information requested.

Approach Considerations:
1. Begin with the core table(s) that contain the
necessary data.
2. Evaluate if additional data from other tables is
absolutely required to fulfill the query's
objective.
3. Construct the query to be as straightforward as
possible, ensuring each step is essential for the
desired outcome.

End Goal:
The primary focus is to develop a query that
efficiently and accurately addresses the question,
emphasizing simplicity and directness in data
retrieval.
```

And the prompt used for error correction is:

```
When executing the SQL query below, some errors
occurred, please fix up SQL based on query and
database info.
When you find an answer, verify the answer
carefully. Include verifiable evidence in your
response if possible.
```

```
When generating SQL, always consider the
constraints:
- In `SELECT <column>`, just select needed columns
in the [Question] without any unnecessary column or
value
- In `FROM <table>` or `JOIN <table>`, do not
include unnecessary table
- If use max or min func, `JOIN <table>` FIRST,
THEN use `SELECT MAX(<column>)` or `SELECT
MIN(<column>)`
- If [Value examples] of <column> has 'None' or
None, use `JOIN <table>` or `WHERE <column> is NOT
NULL` is better
- If use `ORDER BY <column> ASC|DESC`, add `GROUP
BY <column>` before to select distinct values

Schema:
{schema}
Question: {question}
Evidence: {evidence}
SQL query: ```sql
{incorrect_sql}
```
Error: {sqlite_error}

Now please fix the SQL query.

Let's think step by step:
```
```

Appendix B

Error Analysis Examples

Examples from each category in the error analysis is shown in this Appendix. For each category, one example is shown from Mixtral FT with Beam Search, and one from Claude Opus.

Table B.1: Examples from the Dirty Database Values category in the Error Analysis.

	Mixtral	Claude Opus
Database	thrombosis_prediction	california_schools
Utterance	What is the disease patient '30609' diagnosed with. List all the date of laboratory tests done for this patient.	If there are any, what are the websites address of the schools with a free meal count of 1,900-2,000 to students aged 5-17? Include the name of the school.
Evidence	'30609' is an ID; disease means diagnosis	-
Gold query	`SELECT T1.Diagnosis, T2.Date FROM Patient AS T1 INNER JOIN Laboratory AS T2 ON T1.ID = T2.ID WHERE T1.ID = 30609`	`SELECT T2.Website, T1.`School Name` FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.`Free Meal Count (Ages 5-17)` BETWEEN 1900 AND 2000 AND T2.Website IS NOT NULL`
Predicted query	`SELECT Examination.Diagnosis, Laboratory.`Date` FROM Patient INNER JOIN Examination ON Patient.ID = Examination.ID INNER JOIN Laboratory ON Patient.ID = Laboratory.ID WHERE Patient.ID = 30609`	`SELECT schools.School, schools.Website FROM frpm JOIN schools ON frpm.CDSCode = schools.CDSCode WHERE frpm.[Free Meal Count (Ages 5-17)] BETWEEN 1900 AND 2000;`
Explanation	The extra join via the `Examination` table does not contain a value for the requested patient ID.	The predicted query returns one extra row compared to the gold query. This row has a null value for the website.

Table B.2: Example from the Exceptions category in the Error Analysis.

	Mixtral	Claude Opus
Database	california_schools	financial
Utterance	When did the first-through-twelfth-grade school with the largest enrollment open?	Among the accounts who have loan validity more than 24 months, list out the accounts that have the lowest approved amount and have account opening date before 1997.
Evidence	K-12 means First-through-twelfth-grade	-
Gold query	`SELECT T2.OpenDate FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode ORDER BY T1.`Enrollment (K-12)` ASC LIMIT 1`	`SELECT T1.account_id FROM loan AS T1 INNER JOIN account AS T2 ON T1.account_id = T2.account_id WHERE T1.duration > 24 AND STRFTIME('%Y', T2.date) < '1997' ORDER BY T1.amount ASC LIMIT 1`
Predicted query	`SELECT schools.OpenDate FROM schools INNER JOIN frpm ON schools.CDSCode = frpm.CDSCode WHERE frpm.LowGrade = 'K' AND frpm.HighGrade = '12' ORDER BY frpm.Enrollment (K-12) ASC LIMIT 1`	`SELECT account.account_id, amount, duration, date FROM loan JOIN account ON loan.account_id = account.account_id WHERE duration > 24 AND date < '1997-01-01' AND amount = ( SELECT MIN(amount) FROM loan JOIN account ON loan.account_id = account.account_id WHERE duration > 24 AND date < '1997-01-01' );`
Explanation	The predicted query is missing quotation marks around the row `Enrollment (K-12)`.	The date column is ambiguous, as it exist both in the `loan` and `account` table.

Table B.3: Examples from the Incorrect Gold category in the Error Analysis.

Database	superhero	card_games
Utterance	Which hero was the fastest?	How many artists have designed a card with a black border color and is available in both ärenaänd ṁtgoṗrinting type?
Evidence	which hero refers to superhero_name; fastest refers to MAX(attribute_value) WHERE attribute_name = 'Speed';	available in both "arena" and "mtgo" refers to availability like '%arena,mtgo%'
Gold query	`SELECT T1.superhero_name FROM superhero AS T1 INNER JOIN hero_attribute AS T2 ON T1.id = T2.hero_id INNER JOIN attribute AS T3 ON T2.attribute_id = T3.id WHERE T3.attribute_name = 'Speed' ORDER BY T2.attribute_value ASC LIMIT 1`	`SELECT COUNT(CASE WHEN availability LIKE '%arena,mtgo%' THEN 1 ELSE NULL END) FROM cards`
Explanation	There are multiple heroes with Speed attribute at 100, which is max. All of them should be listed as they all answer the utterance.	The black border color condition in the utterance is missing in the gold query.

Table B.4: Example from the Missing Information category in the Error Analysis.

	Mixtral	Claude Opus
Database	formula_1	codebase_community
Utterance	Please list the exact dates on which a Formula_1 race took place on the Barcelona-Catalunya circuit.	How many users received commentator badges in 2014?
Evidence	-	commentator is the name of the badge; year(Date) = 2014;
Gold query	`SELECT T2.date FROM circuits AS T1 INNER JOIN races AS T2 ON T2.circuitID = T1.circuitId WHERE T1.name = 'Circuit de Barcelona-Catalunya'`	`SELECT COUNT(Id) FROM badges WHERE Name = 'Commentator' AND STRFTIME('%Y', Date) = '2014'`
Predicted query	`SELECT date FROM races WHERE circuitId = ( SELECT circuitId FROM circuits WHERE name = 'Barcelona-Catalunya' );`	`SELECT COUNT(UserId) FROM badges WHERE Name = 'commentator' AND STRFTIME('%Y', Date) = '2014'`
Explanation	The utterance does not specify that the circuit is named `Circuit de Barcelona-Catalunya` and not just `Barcelona-Catalunya`.	The evidence specifies that the name should be `commentator`, but the correct name is `Commentator`.

Table B.5: Examples from the Misunderstand category in the Error Analysis.

	Mixtral	Claude Opus
Database	thrombosis_prediction	superhero
Utterance	How many female patients born in 1964 were admitted to the hospital? List them by ID.	Among the superheroes with the race of god/eternal, how many of them are male
Evidence	female refers to SEX = 'F'; born in 1964 refers to YEAR(Birthday) = 1964; admitted to the hospital refers to Admission = '+'	race "god/eternal" refers to race_id = 21; male refers to gender.id = 1
Gold query	`SELECT ID FROM Patient WHERE STRFTIME('%Y', Birthday) = '1964' AND SEX = 'F' AND Admission = '+'`	`SELECT COUNT(*) FROM superhero AS T1 INNER JOIN race AS T2 ON T1.race_id = T2.id INNER JOIN gender AS T3 ON T3.id = T1.gender_id WHERE T1.race_id = 21 AND T1.gender_id = 1`
Predicted query	`SELECT COUNT(ID) FROM Patient WHERE SEX = 'F' AND Birthday LIKE '1964%' AND Admission = '+'`	`SELECT COUNT(*) AS male_god_eternal_count FROM superhero WHERE race_id = (SELECT id FROM race WHERE race = 'god/eternal') AND gender_id = (SELECT id FROM gender WHERE gender = 'male');`
Explanation	The predicted query does not list the patients by ID, as asked in the utterance.	The predicted query tries to match gender with 'male', but the evidence provided states that the male gender is specified by `gender.id = 1`.

Table B.6: Example from the Other Errors category in the Error Analysis.

	Mixtral	Claude Opus
Database	thrombosis_prediction	financial
Utterance	What was the age of the youngest patient when they initially arrived at the hospital?	What is the disposition id of the client who made 5100 USD transaction in 1998/9/2?
Evidence	age refers to SUBTRACT(YEAR(Birthday), YEAR('First Date'))	-
Gold query	`SELECT MIN(STRFTIME('%Y', 'First Date') - STRFTIME('%Y', Birthday)) FROM Patient`	`SELECT T1.disp_id FROM disp AS T1 INNER JOIN trans AS T2 ON T1.account_id = T2.account_id WHERE T2.date = '1998-09-02' AND T2.amount = 5100`
Predicted query	`SELECT 1900 + (strftime('%Y', 'First Date') - strftime('%Y', Birthday)) - (strftime('%m', 'First Date') < strftime('%m', Birthday)) AS age FROM Patient ORDER BY age LIMIT 1`	`SELECT disp_id FROM disp WHERE account_id = ( SELECT account_id FROM trans WHERE amount = 5100 AND date = '1998-09-02'`
Explanation	The predicted query adds `1900`, which there is no reason to do given the utterance, evidence and schema. And it also adds a conditional in the `SELECT` statement that is not grounded in the utterance or evidence.	The model lacks in SQL understanding, `account_id = (SELECT ...)` should be replaced by `account_id IN (SELECT ...)` to get the correct result. The `=` operator considers only one value, and `IN` considers all values returned by the subquery.

Table B.7: Examples from the Semantic Correct category in the Error Analysis.

	Mixtral	Claude Opus
Database	codebase_community	card_games
Utterance	Which user has a higher reputation, Harlan or Jarrod Dixon?	Did the set of cards with ”Angel of Mercy” appear on Magic: The Gathering Online?
Evidence	”Harlan” and ”Jarrod Dixon” are both DisplayName; highest reputation refers to Max(Reputation)	card set ”Angel of Mercy” refers to name = ’Angel of Mercy’; appear on Magic: The Gathering Online refers to mtgoCode is NOT NULL and vice versa
Gold query	`SELECT DisplayName FROM users WHERE DisplayName IN ('Harlan', 'Jarrod Dixon') AND Reputation = ( SELECT MAX(Reputation) FROM users WHERE DisplayName IN ('Harlan', 'Jarrod Dixon') )`	`SELECT IIF(T2.mtgoCode IS NOT NULL, 'YES', 'NO') FROM cards AS T1 INNER JOIN sets AS T2 ON T2.code = T1.setCode WHERE T1.name = 'Angel of Mercy'`
Predicted query	`SELECT DisplayName, Reputation FROM users WHERE DisplayName IN ('Harlan', 'Jarrod Dixon') ORDER BY Reputation ASC LIMIT 1;`	`SELECT CASE WHEN cards.mtgoId IS NOT NULL THEN 'Yes' ELSE 'No' END AS result FROM cards INNER JOIN sets ON cards.setCode = sets.code WHERE cards.name = 'Angel of Mercy'`
Explanation	The predicted query returns both the name of the user and their reputation, while the gold query only return the display name. Therefore, both queries answer the utterance.	Both predicted and gold query correctly answers the utterance. However, `Yes` and `YES` is not considered equal in the evaluation script.

TRITA-EECS-EX-2024:440
Stockholm, Sweden 2024

www.kth.se