# Chapter 1 Introduction

Using natural language to search databases is one of the oldest applications of [Natural Language Processing (NLP)](#_bookmark20) [[1](#_bookmark200)]. The introduction of [Large Language](#_bookmark17) [Models (LLMs)](#_bookmark17) have drastically improved the ability to transform natural language queries into executable [Structured Query Language (SQL)](#_bookmark29). The task of converting natural language into [SQL](#_bookmark29) is known as Text-to-[SQL](#_bookmark29). 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](#_bookmark201)]. [SQL](#_bookmark29) 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](#_bookmark29) aims to add a another layer of abstraction, where the language used to request data is natural language instead of [SQL](#_bookmark29), allowing anyone to retrieve data from a database. The natural language question is converted into [SQL](#_bookmark29) by a language model that knows the structure of the database. Finally, the generated [SQL](#_bookmark29) 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](#_bookmark29) query which when queried against a database answer the natural language question.

Good Text-to-[SQL](#_bookmark29) 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](#_bookmark17) for Text-to-[SQL](#_bookmark29), the challenge of accurately interpreting and converting natural language to [SQL](#_bookmark29) is far from solved. The

current [State-Of-The-Art (SOTA)](#_bookmark28) models are still far from human expert performance [[3](#_bookmark202), [4](#_bookmark203), [5](#_bookmark204)] and cannot reliably be used without carefully reviewing the generated [SQL](#_bookmark29). A short introduction to how [SQL](#_bookmark29) works is provided in [Section 2.1](#_bookmark41).

The current [SOTA](#_bookmark28) model used for Text-to-[SQL](#_bookmark29) is GPT-4 [[6](#_bookmark205)], with various techniques to ensure coherent [SQL](#_bookmark29) generation [[4](#_bookmark203), [5](#_bookmark204), [7](#_bookmark206)]. 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](#_bookmark29). 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](#_bookmark207), [9](#_bookmark208)]. This raises the question if fine-tuned open-source models can outperform the best proprietary [LLMs](#_bookmark17) at the Text-to-[SQL](#_bookmark29) task.

## Research Question

This thesis aims to compare fine-tuning of a *smaller* [LLM](#_bookmark17) and the proprietary language model Claude Opus. A *smaller* [LLM](#_bookmark17) is considered as one which can be fine-tuned on a [Graphics Processing Unit (GPU)](#_bookmark15) limited to using 80GB of [Video Random Access Memory (VRAM)](#_bookmark31). And the selection of proprietary language model is motivated in [Section 4.2.1](#_bookmark133)

**Research Question**: Is there a difference in performance when comparing a

*smaller* fine-tuned [LLM](#_bookmark17) and Claude Opus on the Text-to-[SQL](#_bookmark29) task?

### Hypothesis

Null Hypothesis *H*0: There is no significant differences between the models. Alternative Hypothesis *Ha*: There is a significant difference between the two models.

## Limitations

The main limitation is with respect to compute, fine-tuning [LLMs](#_bookmark17) is a compute intensive task, which require expensive [GPU](#_bookmark15) hardware. The company have offered to provide computational cloud resources for fine-tuning on an A100 [GPU](#_bookmark15) with 80GB [VRAM](#_bookmark31).

The study is limited to only using one model for fine-tuning and only one proprietary [LLM](#_bookmark17) on only one dataset. Furthermore, only having [Application](#_bookmark0) [Programming Interface (API)](#_bookmark0) access for the proprietary [LLM](#_bookmark17) 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](#_bookmark29), what Text-to-[SQL](#_bookmark29) is and various techniques used for Text- to-[SQL](#_bookmark29). Furthermore, [LLMs](#_bookmark17) and the Transformer architecture is covered to help understand how the current models used in Text-to-[SQL](#_bookmark29) work. Some techniques to increase inference speed and decrease training requirements for [LLM](#_bookmark17), used in the thesis, are then covered. Finally, the metrics used for Text- to-[SQL](#_bookmark29) and relevant datasets are presented.

## Databases & SQL

To store data on disk in a structured way, databases are used. These databases are managed by [Database Management Systems (DBMSs)](#_bookmark7), which has been used since the 1960s [[10](#_bookmark209)]. There are a few such systems popular to this day, one of them being the [Structured Query Language (SQL)](#_bookmark29) which is a programming language used to retrieve data from a database. [SQL](#_bookmark29) uses the relational database model introduced by [[2](#_bookmark201)]. 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](#_bookmark29), 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](#_bookmark29) 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](#_bookmark2) dataset, that is also used in most Text-to-[SQL](#_bookmark29) datasets.

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](#_bookmark29) 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](https://www.sqlite.org/lang.html).

## Natural Language and Database query- ing

Using natural language to search databases is one of the oldest applications of [NLP](#_bookmark20). Methods such as using a restricted subset of natural language with explicit rules to convert the text into database queries [[1](#_bookmark200)], and specifying relationships between data and categories [[11](#_bookmark210)] 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](#_bookmark211)]. The current [SOTA](#_bookmark28) models for interacting with databases using natural language use the transformer architecture. [LLMs](#_bookmark17), together with various techniques to represent the database to the model and generate the [SQL](#_bookmark29) has improved the performance of Text-to-[SQL](#_bookmark29) a lot in recent years.

The Text-to-[SQL](#_bookmark29) task consists of predicting an [SQL](#_bookmark29) 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](#_bookmark29). 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.

### 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](#_bookmark200), [11](#_bookmark210)]. But now, there are general methods to create a knowledge representation regardless of the database structure. These methods will be presented in this section.

#### 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](#_bookmark17) has been shown to perform worse at longer contexts [[13](#_bookmark212)], 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](#_bookmark17) is to prompt the [LLM](#_bookmark17) in order to filter relevant tables and columns.

#### 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](#_bookmark213)] was the first [Bidirectional Long Short-Term Memory (BiLSTM)](#_bookmark1) model to do this. It used the WikiSQL [[15](#_bookmark214)] dataset, which only contains one table per database. The names of the columns are passed to the [BiLSTM](#_bookmark1) together with the question.

SyntaxSQLNet [[16](#_bookmark215)] also uses an [BiLSTM](#_bookmark1). 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](#_bookmark29) CREATE TABLE statements ([Data Definition Language](#_bookmark8) [(DDL)](#_bookmark8) Format) or using a Compact Notation. The two formats are shown in [Table 2.1](#_bookmark50). The [DDL](#_bookmark8) 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](#_bookmark8) 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](#_bookmark8) Format and Compact Notation format compared on the same example table. SQLite [DDL](#_bookmark8) Format is used as [DDL](#_bookmark8) format.

#### [DDL](#_bookmark8) Format Compact Notation

CREATE TABLE Employee ( EmployeeID INTEGER NOT NULL,

FirstName TEXT, LastName TEXT, Position TEXT, GroupID INTEGER,

PRIMARY KEY (EmployeeID), FOREIGN KEY (GroupID)

REFERENCES Group(GroupID)

Employee( EmployeeID, FirstName, LastName, Position, GroupID

)

);

Explicit schema definition with data types, constraints and keys.

Compact summary of the schema with only table and column names.

#### 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](#_bookmark216)]. 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](#_bookmark217)], 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)](#_bookmark13) such as in [[19](#_bookmark218)]. Most methods using relational graphs use a relational-aware transformer, also considered a [GNN](#_bookmark13). For the transformer to correctly use the graph as input, relational-aware self-attention is used [[20](#_bookmark219)]. An in depth introduction of normal attention is covered in [Section 2.7](#_bookmark70). This method is also used in various Text- to-[SQL](#_bookmark29) methods [[17](#_bookmark216), [21](#_bookmark220), [22](#_bookmark221), [23](#_bookmark222)]. 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](#_bookmark72). Each position, *ri,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](#_bookmark216), [21](#_bookmark220)], but also be combined with an existing pre-trained transformer [[22](#_bookmark221), [23](#_bookmark222)]. 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](#_bookmark221)] and Graphix-T5 [[23](#_bookmark222)] 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)](#_bookmark21) methods, both the additional weights and the pre-trained transformer weights must be trained together to get good results.

### Output generation techniques

There exist a variety of techniques for generating [SQL](#_bookmark29) queries with language models. The methods using [BiLSTMs](#_bookmark1) 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](#_bookmark17) have adopted some of the techniques used in [BiLSTMs](#_bookmark1), as well as implemented

techniques shown to increase the performance of transformers, such as Beam Search. Some completely new generation techniques with [LLMs](#_bookmark17) for the Text- to-[SQL](#_bookmark29) task have also been developed, for example PICARD [[24](#_bookmark223)].

#### SQL Sketch

The [SQL](#_bookmark29) Sketch was first introduced by SQLNet [[14](#_bookmark213)]. It allows for the model to only predict relevant keywords in the [SQL](#_bookmark29) query, instead of both predicting the grammar and keywords when generating [SQL](#_bookmark29). This is done by first writing a general layout of the [SQL](#_bookmark29) 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](#_bookmark213)] 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](#_bookmark214)] dataset, where each database only contain one table. However, [[14](#_bookmark213)] states that the sketch approach is generalizable to include all possible [SQL](#_bookmark29) queries.

#### SQL Grammar and Syntax Trees

One way to restrict generated output to valid [SQL](#_bookmark29) is to use grammars and abstract syntax trees. Grammar and Syntax Tree based methods design a grammar for the [SQL](#_bookmark29) language. The methods are used in both [BiLSTM](#_bookmark1) [[16](#_bookmark215), [17](#_bookmark216), [25](#_bookmark224)] models and [LLMs](#_bookmark17) [[26](#_bookmark225)].

However, restricting generation to the grammar is insuﬀicient for generating valid [SQL](#_bookmark29), as there are some logical constraints on top of the grammar- decoding process [[25](#_bookmark224)]. Furthermore, to generate valid [SQL](#_bookmark29) 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](#_bookmark29) methods that use a grammar-based generation also apply these methods to ensure that valid [SQL](#_bookmark29) is generated. The [SQL](#_bookmark29) grammar methods can also be used together with other techniques such as Beam Search [[24](#_bookmark223), [26](#_bookmark225)].

#### Beam Search

Beam Search is a method that is widely adopted in [LLMs](#_bookmark17) [[27](#_bookmark226), [28](#_bookmark227), [29](#_bookmark228), [30](#_bookmark229)] and has also been used in [LLMs](#_bookmark17) for the Text-to-[SQL](#_bookmark29) task [[23](#_bookmark222), [24](#_bookmark223), [26](#_bookmark225), [31](#_bookmark230)].

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](#_bookmark17), 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](#_bookmark29) task, there may be further checks such as in PICARD [[24](#_bookmark223)], which removes some of the k selected tokens where the generated [SQL](#_bookmark29) would be invalid.

In [[32](#_bookmark231)], 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](#_bookmark232)]. On the contrary, PICARD [[24](#_bookmark223)] 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](#_bookmark29) from being generated.

#### Self-Correction

A common prompting technique used with [LLMs](#_bookmark17) 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](#_bookmark29) [[4](#_bookmark203), [7](#_bookmark206), [34](#_bookmark233)]. In general, the technique starts with an incorrect [SQL](#_bookmark29) query and prompts the [LLM](#_bookmark17) 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](#_bookmark29) query is generated based on the prompt. If it is incorrect again, the same procedure is repeated for a limited number of tries.

#### Self-Consistency

[LLMs](#_bookmark17) 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](#_bookmark17) 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](#_bookmark234)] uses stochastic sampling together with [Few-Shot (FS)](#_bookmark11) [Chain of Thought (CoT)](#_bookmark4) [[36](#_bookmark235)] prompting.

[CoT](#_bookmark4) is a method that encourages the model to reason before outputting the final answer. This is usually done by giving [FS](#_bookmark11) examples where the answers contain reasoning steps, encouraging the model to generate reasoning steps as well. This method is shown to improve [LLMs](#_bookmark17) performances on tasks which may require reasoning before giving a final answer to the question [[36](#_bookmark235)], for example in mathematics and programming.

Multiple successful Text-to-[SQL](#_bookmark29) methods have drawn inspiration from the self-consistency method [[37](#_bookmark236), [38](#_bookmark237)], with the main difference being that the execution result from executing the generated [SQL](#_bookmark29) outputs are compared, instead of the queries themselves. This is done since two different [SQL](#_bookmark29) queries may yield the same result and carry the same semantic meaning.

## Large Language Models

As shown in [Section 2.2](#_bookmark44), recent methods in Text-to-[SQL](#_bookmark29) use transformer- based [LLMs](#_bookmark17). [LLMs](#_bookmark17) 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](#_bookmark17) have been shown to perform and generalize well, and current models [[6](#_bookmark205), [39](#_bookmark238)] are on par with human experts on a wide variety of professional and academic exams.

The [Generative Pre-trained Transformer (GPT)](#_bookmark14) series of models [[6](#_bookmark205), [40](#_bookmark239), [41](#_bookmark240), [42](#_bookmark241)] 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](#_bookmark240)] was the first successful language model to generalize across tasks and at the time outperform [SOTA](#_bookmark28) specific-purpose models.

To understand how [LLMs](#_bookmark17) 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.

## Tokenization

Tokenization is together with word embeddings the two steps required to transform text to vectors of numbers that a [LLM](#_bookmark17) 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)](#_bookmark3) and WordPiece tokenization. These methods are used for tokenization in [LLMs](#_bookmark17) 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](#_bookmark242)] is to tokenize the training corpus into as few tokens as possible, using at most *K* word or subword tokens [[44](#_bookmark243)]. 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](#_bookmark65) is selected as a new token.

score = freq\_of\_pair freq\_of\_first\_element freq\_of\_second\_element

×

(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 (\_bi*,* ##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](#_bookmark244)] uses the WordPiece tokenization technique.

The other popular technique in transformer models are [BPE](#_bookmark3), originally introduced as a data compression algorithm [[46](#_bookmark245), [47](#_bookmark246)]. 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](#_bookmark3) 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](#_bookmark3) is similar to WordPiece, except that the UNK token doesn’t replace the entire word, but instead only the unknown subword. [BPE](#_bookmark3) is used in models such as the [GPT](#_bookmark14) models [[40](#_bookmark239), [41](#_bookmark240), [42](#_bookmark241)] and BART [[48](#_bookmark247)].

## Word Embeddings

Word embeddings represent tokens in vector space in order to help learning algorithms understand natural language [[49](#_bookmark248)]. 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](#_bookmark248)] and GloVe [[50](#_bookmark249)] 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](#_bookmark75), where the embeddings are learned via backpropagation, and get updated like any other weights in a neural network.

## 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](#_bookmark250)], a sinusoidal positional encoding is used:

PositionalEncoding(*pos,*2*i*) PositionalEncoding(*pos,*2*i*+1)

= sin *pos* (2.2)

100002*i*/*d*embedding

( )

( )

= cos *pos* (2.3)

100002*i*/*d*embedding

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 PositionalEncoding*pos*+*k,i* can be represented as a linear function of PositionalEncoding*pos,i* [[51](#_bookmark250)]. 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](#_bookmark250)]. However, it has also been shown that no positional encoding or embedding layer is required in auto-regressive transformers [[52](#_bookmark251)].

There also exist newer techniques which use positional embeddings: [[53](#_bookmark252), [54](#_bookmark253), [55](#_bookmark254)], which show slight improvements over the original sinusoidal encoding. For example, [Rotary Positional Embeddings (RoPE)](#_bookmark25) [[54](#_bookmark253)] attempts to encode information such that only the relative position between any two tokens is kept when attention is applied. And [[53](#_bookmark252)] tests a Gaussian embedding, utilizing the normal distribution instead of the sinusoidal function.

## Attention

The main building block of the Transformer architecture, which is used in almost all [LLMs](#_bookmark17) is attention. However, attention was initially developed to allow for a [Recurrent Neural Network (RNN)](#_bookmark26) to automatically search the source sentence relevant for predicting the next word, a mechanism known as attention is introduced [[56](#_bookmark255)]. In the context of [NLP](#_bookmark20), 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](#_bookmark255)] used in [RNNs](#_bookmark26) is an additive attention. However, the attention used in [LLMs](#_bookmark17) is a dot-product attention. Scaled dot-product attention [[51](#_bookmark250)] is the main building block of the

transformer architecture. The scaled dot-product attention is defined by:

Attention(*Q, K, V, M* ) =

Softmax

Softmax (*QKT* + *M* )

*exij*

√*dkey*

*V*

∑

(2.4)

(2.5)

(*xij*) =

*k*

*exik*

where *Q* R*T×dkey* , *K* R*T×dkey* and *V* R*T×dvalue* are called Query, Key and Value, *T* is the number of tokens. *M* R*T×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 *dkey* and *dvalue* dimensional representations.

Multi-Head Attention [[51](#_bookmark250)] 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:

MultiHead(*Q, K, V, M, h*) = Concat(head1*,* head2*, . . . ,* head*h*)*WO* (2.6) where head*i* = Attention(*QWQ, KWK, V WV , M* ) (2.7)

*i i* *i*

where *h* represent the number of attention heads, and the attention heads are concatenated into dimensionality R*T×hdvalue* . And *WQ* ∈ R*d*model*×dkey* , *WK* ∈

*i*

*i*

R*d*model*×dkey* , *WV* ∈ R*d*model*×dvalue* , *WO* ∈ R*hdvalue×d*model are learnable weight

*i*

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](#_bookmark256)].

Focusing on how one attention head works, the Key, Query and Value used in [Equation 2.4](#_bookmark72) 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 *QKT* 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 *dvalue* dimension to the larger *dmodel* 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 *dkey* is a scale factor added for numerical stability when running backpropagation through the attention function. In [[51](#_bookmark250)], 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.

## The Transformer Architecture

The Transformer is the underlying architecture used in almost all [LLMs](#_bookmark17), and have been shown to generalize better than specific-purpose models across a variety of tasks [[6](#_bookmark205), [41](#_bookmark240)]. In this section the transformer architecture will be introduced in detail, covering the encoder and decoder blocks in detail as defined by [[51](#_bookmark250)]. And finally, connecting the components together and presenting the entire transformer architecture.

The transformer [[51](#_bookmark250)] 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](#_bookmark3), and the tokens are then converted into learned vector embeddings. At this stage, each vector is of size *d*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.

### 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](#_bookmark70). Before defining the encoder block mathematically, some definitions must be introduced:

The embedded and positionally encoded input vectors to the encoder are denoted as **enc**in R*T*enc*×d*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](#_bookmark73). The Layer Normalization operation is defined as:

( *x* − *µ* )

LayerNorm(*x*) = *γ* ·

√*σ*2 + *ϵ*

+ *β* (2.8)

where *γ, β* R*d*model are learnable parameters, *ϵ* 1 ensures numerical stability, and *µ, σ* R*T*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:

**enc**1 = LayerNorm (**enc**in + MultiHead (**enc**in*,* **enc**in*,* **enc**in*,* **0***, h*)) (2.9)

**enc**out = LayerNorm (**enc**1 + FFN (**enc**1)) (2.10)

where FFN(·) denotes the feed-forward network defined as ReLU(0*, xW*1 + *b*1)*W*2 + *b*2, with weight matrices *W*1 ∈ R*d*model*×dff* , *W*2 ∈ R*dff ×d*model , and bias vectors *b*1 ∈ R*dff* , *b*2 ∈ R*d*model being learnable parameters.

### 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 **dec**in R*T*dec*×d*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 T*dec*×T*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:

∈ R

 

0 −∞ · · · −∞

 

0 0 · · · −∞

*M* =  

(2.11)

 



0 0 · · · 0

Using the defined functions, the decoder is defined by:

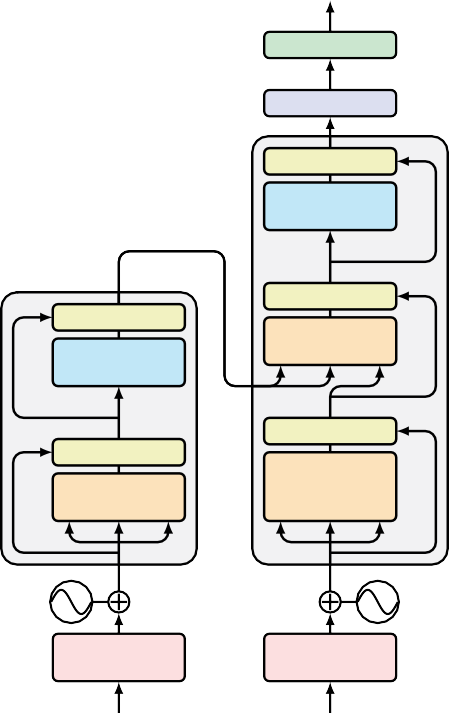
**dec**1 = LayerNorm(**dec**in + MultiHead(**dec**in*,* **dec**in*,* **dec**in*, M, h*)) (2.12) **dec**2 = LayerNorm(**dec**1 + MultiHead(**dec**1*,* **enc**out*,* **enc**out*,* **0***, h*)) (2.13) **dec**3 = LayerNorm(**dec**2 + FFN(**dec**2)) (2.14)

where FFN(·) is defined as in the encoder.

### End-to-End Functionality of the Transformer Architecture

Finally, the encoder and decoder blocks are stacked *N* = 6 times[[51](#_bookmark250)]. The encoder blocks are sequentially connected by feeding encout from the one block as input to the next block. The encoder output from the last block is

Output Probabilities

Softmax Linear

Add & Norm

Feed Forward

Add & Norm Feed

Add & Norm Multi-Head

Attention *N* ×

Forward V K Q

*N*

×

Add & Norm

Multi-Head Attention

Add & Norm

Masked Multi-Head Attention

Positional Encoding

V K Q

V K Q

Positional Encoding

Input Embedding

Output Embedding

Inputs Outputs

Figure 2.1: The Transformer architecture, recreated based on [[51](#_bookmark250)], 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 decout 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](#_bookmark83).

## Transfer Learning

Training [LLMs](#_bookmark17) from scratch is a compute intensive task that may take months on large [GPU](#_bookmark15)-clusters and require terabytes of training data [[58](#_bookmark257)]. Hence, some datasets are too small to use when training [LLMs](#_bookmark17) 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)](#_bookmark12) on the target domain [[59](#_bookmark258)]. 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](#_bookmark17), it is common to fine-tune models for specific tasks such as chatting [[6](#_bookmark205)], generating code in a programming language [[60](#_bookmark259)] and more.

## 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](#_bookmark31) but require at least

1.2TB of [VRAM](#_bookmark31) during training when using the Adam optimizer [[61](#_bookmark260)]. 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](#_bookmark21) includes methods which fine-tune few parameters in comparison to the model size, while still updating parameters for all layers.

### 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](#_bookmark261)] 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](#_bookmark83). 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](#_bookmark261)] show that the [FT](#_bookmark12) models using Adapters perform within 1 percent point compared to fine-tuning the full model on the GLUE [[63](#_bookmark262)] dataset.

AdaMix [[64](#_bookmark263)] takes an approach similar to [Sparse Mixture of Experts (SMoE)](#_bookmark27) in the adapters, but the expert selection is random at each iteration instead of using a gating function. The model tested by [[64](#_bookmark263)] 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.

### Low-Rank Adaptation Fine-Tuning

[Low-Rank Adaptation (LoRA)](#_bookmark18) fine-tuning [[61](#_bookmark260)] 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](#_bookmark264)]. The matrices are assumed to be of a low rank, *r*, indicating that a weight matrix can be factorized as *W* = *ABT* , where *W d×k* and *A, B d×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](#_bookmark18) is done as *W*0*x* + *BAT x*, 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*

∈ R ∈ R

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](#_bookmark12) matrix, *BAT W*0 *Woptimal* acting as the two factors of a low-rank fine-tuneable matrix which is added to the original pre-trained weight.

~ −

Using [LoRA](#_bookmark18) reduces the number of gradients to be updated, which decreases the memory usage for gradients and optimizer values. Furthermore, [[61](#_bookmark260)] 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](#_bookmark29) benchmark WikiSQL.

QLoRA [[66](#_bookmark265)] is a method that builds upon [LoRA](#_bookmark18), 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](#_bookmark265)] 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](#_bookmark18) [[61](#_bookmark260)]. QLoRA also optimize for memory by paging optimizer values on the regular [Random](#_bookmark23) [Accress Memory (RAM)](#_bookmark23) when the [GPU](#_bookmark15) [VRAM](#_bookmark31) is full in order to avoid out of memory if there are short memory spikes.

### Few-Shot Prompting

Another way to teach [LLMs](#_bookmark17) is through in-context learning, which does not require any fine-tuning. [FS](#_bookmark11) in-context learning is a method which provides examples to the language model during inference [[67](#_bookmark266), [68](#_bookmark267)]. This technique is used to allow pre-trained general-purpose language models to excel in specific tasks, such as Text-to-[SQL](#_bookmark29) [[4](#_bookmark203), [5](#_bookmark204), [7](#_bookmark206), [33](#_bookmark232), [34](#_bookmark233), [37](#_bookmark236)]. However, there are some diﬀiculties with using [FS](#_bookmark11). For example, the ordering of the examples [[68](#_bookmark267)] and the selection of examples [[67](#_bookmark266)] is important for the quality of the generated output.

## Sparse Mixture of Experts

Another method to decrease memory requirements and increase inference speed without any cost of performance is to use a [SMoE](#_bookmark27) architecture. [SMoE](#_bookmark27) 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](#_bookmark27) also includes models in which only a part of the model uses a [Mixture of Experts (MoE)](#_bookmark19) structure. For example, in transformer models, it is common that [SMoE](#_bookmark27) is applied only to the feed- forward network, but not to the multi-head attention [[69](#_bookmark268)]. When applied to the feed-forward network of a transformer, the FFN(·) function in [Equation 2.10](#_bookmark77) and [Equation 2.14](#_bookmark81) is replaced by GatedFFN(·):

*N−*1

∑

GatedFFN(*x*) = *G*(*x*)*i*FFN*i*(*x*) (2.15)

*i*=0

*G*(*x*) = Softmax(TopK(*x* · *Wg*)) (2.16)

where *G*( ) is the gating function, FFN*i*( ) are the experts, *N* the total number of experts and *Wg* R*a×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( ) selects the *K* largest values computed, and sets the rest to , making the Softmax output non- zero values only for *K* experts. The TopK function is what makes the [MoE](#_bookmark19) 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](#_bookmark269)]. However, training a [SMoE](#_bookmark27) model comes with challenges, [[69](#_bookmark268)] finds that [SMoE](#_bookmark27) models are more prone to diverging loss, and most papers reviewed by [[69](#_bookmark268)] had some issues with loss instability. An attempt to mitigate the instability is to add a balancing loss term that penalizes large logits, *x Wg*, such as the router z-loss [[71](#_bookmark270)], 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](#_bookmark270)].

·

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](#_bookmark271)]. 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](#_bookmark272)]. Furthermore, there is no strong correlation between experts and domains, but there is some correlation between experts and language [[72](#_bookmark271)]. For example, has, had and have are routed to the same expert [[72](#_bookmark271)].

Although the mentioned training diﬀiculties, [SMoE](#_bookmark27) models have been shown to work well with [LLMs](#_bookmark17) [[72](#_bookmark271), [73](#_bookmark272)]. The main upside of [SMoE](#_bookmark27) 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](#_bookmark27) with less [VRAM](#_bookmark31) compared to dense models of similar size [[74](#_bookmark273)].

## Evaluation metrics

There are a few different ways to measure how good a model is at the Text-to-[SQL](#_bookmark29) task. The most commonly used metrics are presented in this section.

### Component Matching

Component Matching [[75](#_bookmark274)] divides the [SQL](#_bookmark29) query into different components:

SELECT, WHERE, GROUP BY, ORDER BY and KEYWORDS. KEYWORDS

include all [SQL](#_bookmark29) 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](#_bookmark275), [77](#_bookmark276)] of the components are computed to find the component matching score for an [SQL](#_bookmark29) 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.

### Exact Matching

[Exact Matching Accuracy (EM)](#_bookmark9) [[75](#_bookmark274)] uses the same method as Component Matching to decompose the [SQL](#_bookmark29) query into components. However, the entire decomposed query must match the ground truth query to be considered as correct. This makes [EM](#_bookmark9) a binary metric for each datapoint, unlike component matching that considers partially matching queries as partially correct. To compute the [EM](#_bookmark9) score for an entire set of values, the average is used.

### Execution Accuracy

[Execution Accuracy (EX)](#_bookmark10) [[75](#_bookmark274)] 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](#_bookmark10) is be defined as

∑*N* 1 ˆ

EX(*V, V*ˆ) =  *n*=1 *Vn*=*Vn*

*N*

(2.17)

where *Vn* is the set of rows produced from the gold query and *V*ˆ*n* is the set of rows produced by the predicted query for the nth datapoint [[3](#_bookmark202)].

### Valid Eﬀiciency Score

[Valid Eﬀiciency Score (VES)](#_bookmark30) [[3](#_bookmark202)] is a combination of [EX](#_bookmark10) and relative eﬀiciency between the predicted and gold query. Eﬀiciency can refer to execution time, memory cost, or throughput. However, in [[3](#_bookmark202)] they only use eﬀiciency in terms of execution time. Therefore, [VES](#_bookmark30) will be explained in terms of execution time.

The score compares the execution time of the predicted [SQL](#_bookmark29) query against the gold [SQL](#_bookmark29) query and weighs the [EX](#_bookmark10) according to the time eﬀiciency score. The formal definition of [VES](#_bookmark30) is:

VES(*Y, Y*ˆ) = 1

*N*

*n*

*n*

*E*(*Y*ˆ*n*)

*N*

1

∑

*n*=1

*Vn*=*V*ˆ*n*

· R(*Yn*

*, Y*ˆ*n*) (2.18)

R(*Y*

*, Y*ˆ ) = √*E*(*Yn*)

(2.19)

where *N* is the number of datapoints, *Vn* and *V*ˆ*n* are defined the same way as in [Section 2.12.3](#_bookmark100). *Yn* and *Y*ˆ*n* represent the gold and predicted query, respectively. *E*( ) is a function to measure the execution time. The square root function is

·

applied to dampen the effect of outliers as √*x* 1 *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](#_bookmark30) score is not upper-bounded either. However, assuming that the gold queries are optimally designed in terms of execution speed, the [VES](#_bookmark30) score is upper bounded by 1 as the R-function must be upper bounded by 1 when *Yn* is optimal.

To compute *E*( ) the execution time is estimated by executing the same query multiple times. In [[3](#_bookmark202)], the [SQL](#_bookmark29) query is executed 100 times and the [VES](#_bookmark30) score for *N* samples are computed according to [Algorithm 1](#_bookmark103), 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](#_bookmark29) and computes the time the execution takes.

·

**Algorithm 1** [Valid Eﬀiciency Score](#_bookmark30)

1: **function** VES(*V* , *V*ˆ, *Y* , *Y*ˆ)

2: *V ES* 0

→

3: **for** *n* 1 to *N* **do**

→

4: **if** *Vn* = *V*ˆ*n* **then**

5: differences → {}

6: **for** *i* → 1 to 100 **do** ˆ

7: predicted\_time execute\_sql(*Yn*)

→

8: gold\_time execute\_sql(*Yn*)

→

9: time\_fraction predicted\_time/gold\_time

→

10: append time\_fraction to differences

11: differences → remove\_outliers(differences)

12: *R*\_*score* → √ ∑(differences)

length(differences)

13: *V ES* → *V ES* + *R*\_*score*

14: *V ES V ES*

*N*

→

15: **return** *V ES*

## Datasets

There exist quite a few Text-to-[SQL](#_bookmark29) datasets [[3](#_bookmark202), [15](#_bookmark214), [75](#_bookmark274), [78](#_bookmark277), [79](#_bookmark278), [80](#_bookmark279), [81](#_bookmark280), [82](#_bookmark281), [83](#_bookmark282), [84](#_bookmark283), [85](#_bookmark284)]. However, only two datasets are widely used for benchmarking [LLMs](#_bookmark17). These datasets are Spider [[75](#_bookmark274)] and [BIRD](#_bookmark2) [[3](#_bookmark202)]. The datasets consist of databases, as well as written natural language questions called utterances paired with ground truth [SQL](#_bookmark29) queries called gold queries. A comparison of the sizes of the Spider and [BIRD](#_bookmark2) datasets is shown in [Table 2.2](#_bookmark106). Both datasets uses the SQLite database language for the databases.

Table 2.2: A comparison of the contents in the Spider and [BIRD](#_bookmark2) datasets.

|  |  |  |
| --- | --- | --- |
| **Metric** | **Spider** | **[BIRD](#_bookmark2)** |
| 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) |

### Spider

Spider [[75](#_bookmark274)] was the first large-scale cross-domain Text-to-[SQL](#_bookmark29) 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](#_bookmark29) tutorials, Databaseanswers[∗](#_bookmark110) and 90 databases from the WikiSQL [[15](#_bookmark214)] dataset. The databases were then populated with fabricated data. After the collection process, questions were created manually, and [SQL](#_bookmark29) queries were created to answer the questions, yielding a total of 10 181 questions answered with 5 693 [SQL](#_bookmark29) queries. However, in the released version of Spider[†](#_bookmark111), six existing datasets are also incorporated in the Spider dataset. The combination of these six datasets and the data created by

[[75](#_bookmark274)] is displayed in [Table 2.2](#_bookmark106).

Spider uses the [EM](#_bookmark9) and [EX](#_bookmark10) scores to evaluate performance of models on their dataset.

### BIRD

[Big Bench for Large-scale Database Grounded in Text-to-SQL Tasks](#_bookmark2) [(BIRD)](#_bookmark2) [[3](#_bookmark202)] 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](#_bookmark29) datasets, making writing eﬀicient [SQL](#_bookmark29) queries of importance. Therefore, the [VES](#_bookmark30) score is introduced and used as a metric of how fast the generated [SQL](#_bookmark29) queries are executed compared to the gold queries. The other metric used to benchmark models on [BIRD](#_bookmark2) is [EX](#_bookmark10).

Each data point consists of an utterance, a gold [SQL](#_bookmark29) 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](#_bookmark113). In the first example, the Reach column and unique

∗[databases.biz](https://databases.biz/)

†[yale-lily.github.io/spider](https://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](#_bookmark29) queries, such as in Example 2 in [Table 2.3](#_bookmark113), 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](#_bookmark2) dataset

Example 1 Example 2

**Utterance**

**Evidence**

**Gold query**

How many unique users have seen tweet with text ”Happy New Year to all those AWS instances of ours!”?

”Happy New Year to all those AWS instances of ours!” is the text; seen unique users refers to Reach

SELECT Reach FROM twitter

WHERE 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?

male user refers to Gender

= ’Male’; ’Monday’ is the Weekday; average number of likes = Divide (Sum(Likes), Count(TweetID))

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](#_bookmark2) is hidden, but models can be evaluated by

contacting the authors for submission[∗](#_bookmark115). 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](#_bookmark17).

∗[bird-bench.github.io](https://bird-bench.github.io/)

# Chapter 3 Related Work

In this section, the recent advancements in Text-to-[SQL](#_bookmark29) 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](#_bookmark2).

## Prompting Strategies

Various prompting strategies are used, but the majority of the [LLM](#_bookmark17) 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](#_bookmark285)] and C3 [[38](#_bookmark237)], 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](#_bookmark206)] 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](#_bookmark29) to be generated:

* Easy: Question, schema link, and answer format with [FS](#_bookmark11). 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](#_bookmark286)], an intermediate representation between natural language and [SQL](#_bookmark29) developed for Text-to-[SQL](#_bookmark29). Both

NatSQL and [SQL](#_bookmark29) 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](#_bookmark29) answer is written. This format is used for [FS](#_bookmark11) 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](#_bookmark29), 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](#_bookmark2) test dataset is MAC-SQL [[4](#_bookmark203)], 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](#_bookmark4) prompting strategy is used as well as providing [FS](#_bookmark11) 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](#_bookmark29) query.

DAIL-SQL [[5](#_bookmark204)] compares various [FS](#_bookmark11) prompting strategies, and unlike the previous methods, they use [Retrieval Augmented Prompting (RAG)](#_bookmark24) to select the [FS](#_bookmark11) 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](#_bookmark222)] model, and then comparing that query against the gold query. For both [FS](#_bookmark11) examples and the final prompts, various prompting strategies are tested. For the [FS](#_bookmark11) examples, three prompts are tested:

* Full information prompt, containing the database schema, utterance, and gold [SQL](#_bookmark29).
* [SQL](#_bookmark29)-Only prompt, containing only the [SQL](#_bookmark29) queries.
* DAIL organization prompt, containing the [SQL](#_bookmark29) 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](#_bookmark8) 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](#_bookmark11) examples.

Another method which uses [RAG](#_bookmark24) manage to outperform many of the methods using GPT-4 [[34](#_bookmark233)]. In SQL-PaLM [[37](#_bookmark236)], [FS](#_bookmark11) and [FT](#_bookmark12) is compared using the PaLM-2[[88](#_bookmark287)] 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](#_bookmark11), and it has been shown that [FS](#_bookmark11) can increase the [EX](#_bookmark10) on Spider and [BIRD](#_bookmark2) 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](#_bookmark12) models, and with PaLM-2, no increase in performance was shown when Fine- Tuning the model.

### Output Generation

The output generation techniques mainly use either Self-Correction or Self- Consistency methods. However, DAIL-SQL [[5](#_bookmark204)] use the generated output from GPT-4 without modifying it, and outperforms most other techniques in terms of [EX](#_bookmark10).

MAC-SQL [[4](#_bookmark203)] uses Self-Correction, and prompt GPT-4 until the generated [SQL](#_bookmark29) no longer contain errors. In [[34](#_bookmark233)], 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](#_bookmark10) 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](#_bookmark29) errors to GPT-4, but rather only provide guidance concerning standard issues that [SQL](#_bookmark29) 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](#_bookmark236)] and C3 [[38](#_bookmark237)] use Self-Consistency, generating multiple outputs for the same prompt and selecting the most commonly generated prompt. In [[38](#_bookmark237)] 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.

## Literature Summary

All recent papers use [LLMs](#_bookmark17), most by developing new prompting strategies rather than fine-tuning the model. The only exception is SQL-PaLM, which fine-tunes PaLM-2 [[88](#_bookmark287)] and finds that the [FT](#_bookmark12) model has a similar performance to the pre-trained model. The papers with reported results on [BIRD](#_bookmark2) and Spider are presented in [Table 3.1](#_bookmark124) and [Table 3.2](#_bookmark125), respectively. Each paper tries different prompting strategies. However, some of the most used strategies are schema linking and [FS](#_bookmark11) 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](#_bookmark2) dataset, with details on knowledge representation and output generation methods. The metrics are reported according to the oﬀicial benchmark scores at [bird-](https://bird-bench.github.io/) [bench.github.io](https://bird-bench.github.io/).

Knowledge Representation

Model

Output Method

[E](#_bookmark10)X ↑ [VE](#_bookmark30)S ↑

Dev Test Dev Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| DIN-SQL+GPT-4 [[7](#_bookmark206)] | Text | Self-Correction | 50.72 | 55.90 | 58.79 | 59.44 |
| DAIL-SQL+GPT-4 [[5](#_bookmark204)] | Text | None | 54.76 | 57.41 | 56.08 | 61.95 |
| MAC-SQL+GPT-4 [[4](#_bookmark203)] | 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-](https://yale-lily.github.io/spider) [lily.github.io/spider](https://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.

Knowledge Representation

Model

Output Method

[EM](#_bookmark9) ↑ [E](#_bookmark10)X ↑

Dev Test Dev Test

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Graphix-T5 [[23](#_bookmark222)] | Relational | PICARD | 77.1 | 74.0 | 81.0\* | 77.6 |
| ChatGPT-SQL [[86](#_bookmark285)] | Text | None | - | - | 70.1\* | - |
| DIN-SQL+GPT-4 [[7](#_bookmark206)] | Text | Self-Correction | 60.1 | 60 | 74.2\* | 85.3 |
| SQL-PaLM [FT](#_bookmark12) [[37](#_bookmark236)] | Text | Self-Consistency | - | - | 82.8\* | - |
| SQL-PaLM [FS](#_bookmark11) [[37](#_bookmark236)] | Text | Self-Consistency | - | - | 82.7\* | - |
| C3+ChatGPT [[38](#_bookmark237)] | Text | Self-Consistency | - | - | 81.8\* | 82.3 |
| DAIL-SQL+GPT-4 [[5](#_bookmark204)] | Text | None | 71.9\* | - | 82.4\* | 86.2 |
| RAG+ChatGPT [[34](#_bookmark233)] | Text | Self-Correction | - | - | 85.0\* | - |
| MAC-SQL+GPT-4 [[4](#_bookmark203)] | 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](#_bookmark17) for [FT](#_bookmark12), 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.

## Dataset & Metrics

The datasets Spider and [BIRD](#_bookmark2) are both of similar size, covering a wide variety of domains and with a similar amount of columns per table. However, the [BIRD](#_bookmark2) 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](#_bookmark29) keywords, such as functions. Therefore, the [BIRD](#_bookmark2) dataset was chosen. And consequently, the evaluation metrics chosen were [EX](#_bookmark10) and [VES](#_bookmark30) to obtain comparable results with prior work, as these are used to benchmark [BIRD](#_bookmark2). However, when analyzing and comparing models, only the [EX](#_bookmark10) score is used. As the training and evaluation data were publicly available, these splits were used for training and evaluating the model.

## 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](#_bookmark31) A100 [GPU](#_bookmark15). And the large pre-trained model was used via an [API](#_bookmark0),

Table 4.1: Performance of closed source [LLMs](#_bookmark17) on the datasets MMLU and HumanEval. The scores are provided by the respective [API](#_bookmark0) 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.

### 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](#_bookmark29), and not require further fine-tuning to be used. In [Chapter 3](#_bookmark116), 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](#_bookmark0) 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](#_bookmark288)] and HumanEval [[90](#_bookmark289)] 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](#_bookmark131), 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](#_bookmark2) use GPT-4. The Claude Opus version used was claude-3-opus-20240229.

### Smaller Fine-tuneable Model

As the model will be [FT](#_bookmark12), 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 modi- fications to using high-quality data to using [SMoE](#_bookmark27). However, all models have in common that they use the decoder-only transformer architecture.

The Llama [[91](#_bookmark290)] and Llama2 [[92](#_bookmark291)] models use normalization before the attention block and before the FFN block, instead of after. Furthermore, SwiGLU [[93](#_bookmark292)] is used as the activation function and Rotary Embeddings [[54](#_bookmark253)] are used instead of the positional embeddings used in [[51](#_bookmark250)]. Both Llama and Llama2 exist in dif- ferent sizes, ranging from 7B to 70B parameters. The data used is open-source, with the majority (67 %) coming from the Common Crawl[∗](#_bookmark136) 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](#_bookmark265), [94](#_bookmark293)]. Orca [[94](#_bookmark293)] 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](#_bookmark265)] introduces QLoRA and uses it to fine-tune the Llama models on the OASST1[†](#_bookmark137) 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](#_bookmark265)].

The Phi-1 [[95](#_bookmark294)] 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](#_bookmark12) Phi-1 model outperforms GPT-3.5 on the HumanEval benchmark, despite being more than 100 times smaller than GPT- 3.5.

Mistral [[96](#_bookmark295)] is a 7B model using sliding-window attention, restricting the mask matrix *M* in [Equation 2.11](#_bookmark80) 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 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](#_bookmark272)] model utilizes a [SMoE](#_bookmark27) architecture as described in [Section 2.11](#_bookmark94). 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 some- where 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](#_bookmark131). 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. Further- more, the [SMoE](#_bookmark27) architecture may help the model excel at Text-to-[SQL](#_bookmark29) 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](#_bookmark12) version of Mixtral was used.

## 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](#_bookmark29). In contrast, the prompt for Claude Opus was de- signed to add instructions to mitigate biases and avoid making common mis- takes. Furthermore, a prompt used for Mixtral without fine-tuning was also de- signed 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](#_bookmark8) format for providing the database schema, without providing the extra column information that exist for some columns by the [BIRD](#_bookmark2) 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](#_bookmark0). The prompts used are presented in [Appendix A](#_bookmark302).

### Claude Opus

The prompt used for Claude Opus is inspired by the existing papers on the [BIRD](#_bookmark2) dataset using GPT-4. The prompt was designed to be cost-eﬀicient, limiting the cost per prompt where necessary. Therefore, [FS](#_bookmark11) 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](#_bookmark2) 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 some- times generates invalid [SQL](#_bookmark29), often due to non-standard column naming con- ventions in the schema which are ignored, and sometimes due to other syn- tactical errors in the [SQL](#_bookmark29). Therefore, it was decided to use a self-correction step. It was found that by simply prompting the [SQL](#_bookmark29) 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](#_bookmark304).

### Mixtral

The prompt used for [FT](#_bookmark12) Mixtral was designed to only contain the necessary information, allowing the fine-tuning process to learn how to generate [SQL](#_bookmark29) 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](#_bookmark29) queries to learn from, making it diﬀicult to teach the model how to fix an incorrect [SQL](#_bookmark29) query. The fine-tuning prompt is available in [Section A.2](#_bookmark305).

Finally, the pre-trained Mixtral baseline without fine-tuning was designed us- ing 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, ut- terance, and evidence together with the error. This prompt is presented in [Section A.3](#_bookmark306).

## 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](#_bookmark2) evaluation script. This in- cludes 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 re- turned, and the generated and gold query returns a correct interpretation of the utterance, it is also considered semantically correct.
* **Misunderstand**: When the [LLM](#_bookmark17) misunderstands the utterance, evi- dence, or provided schema, and therefore generates a query which does not answer the utterance. [SQL](#_bookmark29) queries are considered as misunder- standing the utterance or evidence if there is no attempt to use all the necessary information, and misunderstanding the schema includes mis- interpreting 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](#_bookmark8) 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 in- correct ordering, or a lack of [SQL](#_bookmark29) knowledge. Furthermore, blatantly wrong queries, for example, a query that ignores the utterance com- pletely. 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 “Mis- understand” and these errors is that in a “Misunderstand”, some neces- sary information is used, such as using the correct tables or considering some correct conditions.

Examples of each category from the [BIRD](#_bookmark2) evaluation dataset are provided in [Appendix B](#_bookmark308).

## 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.

### Software

The framework used for fine-tuning and inference with Mixtral was Axolotl[∗](#_bookmark148), which is a framework built on top of HuggingFace to easily fine-tune [LLMs](#_bookmark17). The framework supports both [LoRA](#_bookmark18) 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](#_bookmark0) using Python.

The evaluation code provided by [BIRD](#_bookmark2) is also implemented in Python, and adapted to be used with in my code to evaluate [SQL](#_bookmark29) queries.

The statistical methods presented in [Section 4.6](#_bookmark155) used the Python libraries Scipy and Numpy. Numpy was used to compute the confidence intervals using

∗[github.com/OpenAccess-AI-Collective/axolotl](https://github.com/OpenAccess-AI-Collective/axolotl)

[Equation 4.1](#_bookmark158). P-values were computed using scipy.stats.t.sf with *T*

and *df* computed via [Equation 4.2](#_bookmark160) and [Equation 4.4](#_bookmark162).

### Hardware

To fine-tune Mixtral, the service Paperspace was used, where a machine with an A100 [GPU](#_bookmark15) with 80GB [VRAM](#_bookmark31) was used for both training and inference. The cost of this machine was covered by the company.

### Fine-Tuning

In order to be able to fine-tune Mixtral 8x7B on one 80GB A100 [GPU](#_bookmark15), 4-bit QLoRA quantization was used. Furthermore, the context length was limited to 4096 tokes, which excluded prompts from one database in the [BIRD](#_bookmark2) 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](#_bookmark27) 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](#_bookmark152). The QLoRA hyper- parameters were selected as recommended[[66](#_bookmark265)]. 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](#_bookmark31), but does not have a significant impact on quality [[97](#_bookmark296)].

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 |

### Evaluation

The models will be evaluated against baselines. These include the existing models evaluated on [BIRD](#_bookmark2) presented in [Chapter 3](#_bookmark116), 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](#_bookmark12), Mixtral baseline and Claude Opus were evalu- ated on the [BIRD](#_bookmark2) evaluation dataset, using the respective formats explained in [Section 4.3](#_bookmark139). the [FT](#_bookmark12) 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](#_bookmark29). Furthermore, the generation of the models was set to be deterministic, such that asking the same question always generates the same [SQL](#_bookmark29) 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.

## Statistical Methods

To evaluate the performance of the models, some statistical tools are used. The [FT](#_bookmark12) and pre-trained models are compared against each other using paired testing, and the scores reported for each individual test are the [EX](#_bookmark10) and [VES](#_bookmark30) scores, both reported with 95 % **[C](#_bookmark5)**[onfidence](#_bookmark5) **[I](#_bookmark5)**[ntervals (CIs)](#_bookmark5).

### Confidence Intervals

To show the reliability of each estimate, [CIs](#_bookmark5) are used. The [CIs](#_bookmark5) are computed using a Z-score of 1*.*96, corresponding to a 95 % [CI](#_bookmark5). The formula for comput- ing [CIs](#_bookmark5) is shown in [Equation 4.1](#_bookmark158), where *x*¯ is the sample mean, *s* is the sample standard deviation and *n* is the number of samples.

[CI](#_bookmark5) = *x*¯ ± *Z* √*n* (4.1)

*s*

### Paired testing

To perform paired testing, statistical significance testing is used with a null hypothesis *H*0 : *x*¯1 = *x*¯2 and an alternative hypothesis *Ha* : *x*¯1 = *x*¯2, where *x*¯1 is the mean result using one method and *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 Theo-](#_bookmark6) [rem (CLT)](#_bookmark6) [[98](#_bookmark297)], the mean of a sum of [Independent and Identically Distributed](#_bookmark16) [(IID)](#_bookmark16) 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](#_bookmark298)]. 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](#_bookmark299)]. 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](#_bookmark160) and [Equation 4.3](#_bookmark161), where *t*( ) is the t-distribution and *df* the degrees of freedom. *s* is the sample standard deviation, *x*ˆ is the sample mean and *n* the number of samples.

·

*x*¯1 − *x*¯2

√ *s*

(4.2)

*T* = 2 2

+

*s*

1 2

*n*1 *n*2

*p* = *t*(*df, T* ) (4.3)

(*s*2/*n* + *s*2/*n* )2

*df* = 1 1 2 2

(4.4)

(*s*2/*n* )2 (*s*2/*n* )2

1 1 + 2 2

*n*1*−*1 *n*2*−*1

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](#_bookmark165), the results of Mixtral and Claude Opus is shown. Mixtral [FT](#_bookmark12) with Beam Search and Claude Opus has a statistically insignificant difference in terms of [EX](#_bookmark10) score with a p-value of 0.8005. Furthermore, their [VES](#_bookmark30) is also similar.

Table 5.1: Results of Claude Opus and Mixtral, and the work presented in [Chapter 3](#_bookmark116). [CIs](#_bookmark5) are reported with ± for the evaluated models.

Model [E](#_bookmark10)X ↑ [VE](#_bookmark30)S ↑ DIN-SQL+GPT-4 [[7](#_bookmark206)] 50*.*72 55*.*90

DAIL-SQL+GPT-4 [[5](#_bookmark204)] 54*.*76 57*.*41

MAC-SQL+GPT-4 [[4](#_bookmark203)] 57*.*56 59*.*59

Mixtral [FT](#_bookmark12) + Beam Search 50*.*52 ± 2*.*50 54*.*53 ± 5*.*85

Mixtral [FT](#_bookmark12) 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](#_bookmark10) grouped by diﬀiculty as shown in [Table 5.2](#_bookmark166), both models are also similar in performance. Although Claude Opus has slightly higher overall and grouped scores than Mixtral on all but the Challenging dif- ficulty, the differences are insignificantly different, and it cannot be concluded that one model can be preferred over the other.

Table 5.2: [EX](#_bookmark10) presented for Mixtral [FT](#_bookmark12) 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

## Baseline

Both Mixtral [FT](#_bookmark12) with Beam Search and Claude Opus perform similar to DIN- SQL [[7](#_bookmark206)], but are worse than DAIL-SQL [[5](#_bookmark204)] and MAC-SQL [[4](#_bookmark203)].

Comparing Mixtral [FT](#_bookmark12) and Mixtral [FT](#_bookmark12) 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](#_bookmark28) in terms of [EX](#_bookmark10), and 5 percent points behind in terms of [VES](#_bookmark30).

## 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](#_bookmark170). Over 20 % of the incorrectly classified data had errors in the gold [SQL](#_bookmark29). Outside of the incorrectly classified samples, Mixtral [FT](#_bookmark12) with Beam Search strug- gled the most with misunderstanding the utterance, evidence, or [DDL](#_bookmark8) 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](#_bookmark29), where 68 queries could not be executed due to errors in the [SQL](#_bookmark29), 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.

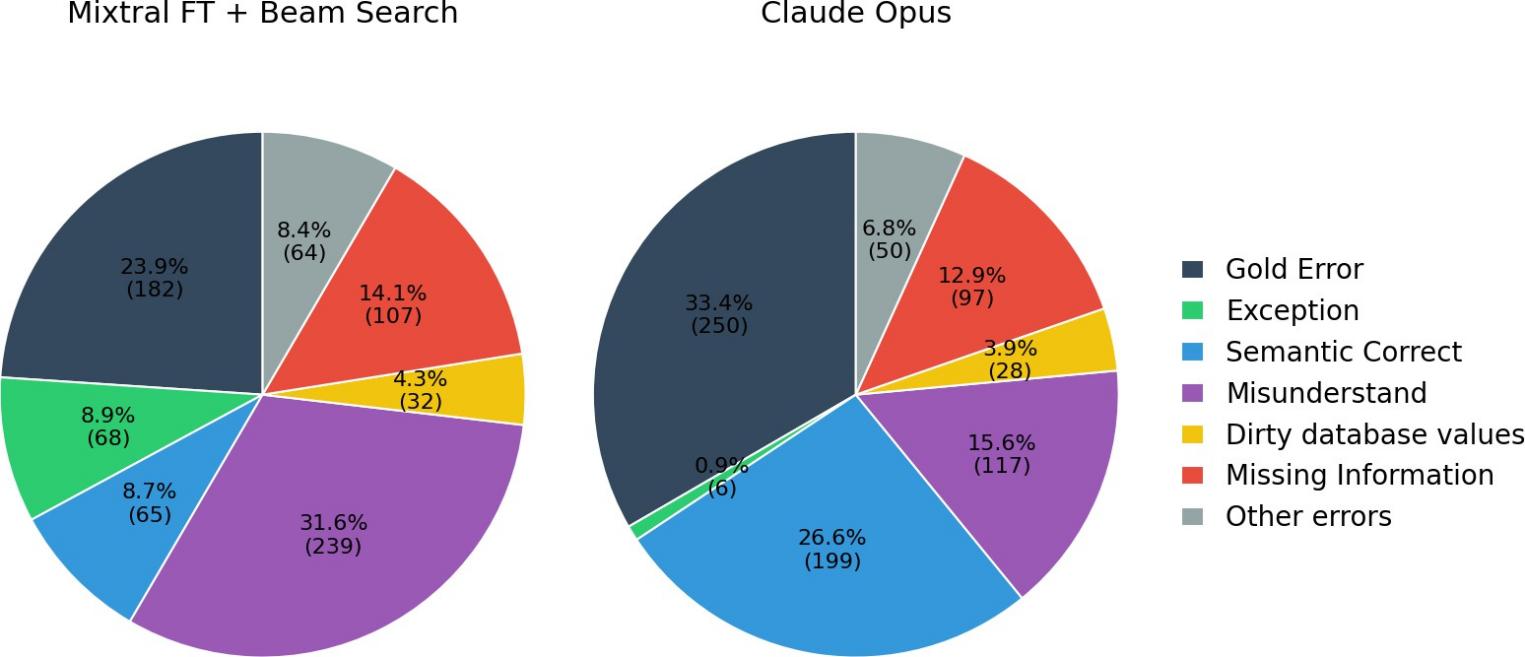


Figure 5.1: The incorrect samples for Mixtral [FT](#_bookmark12) + Beam Search and Claude Opus, sorted into categories based on what caused the mismatch between gold query and predicted query. In total, In total, there are 760 incorrect datapoints for Mixtral and 752 incorrect datapoints for Claude Opus.

With all Semantic Correct queries considered as correct, and removing all Gold Errors and Missing Information from the database, the [EX](#_bookmark10) 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](#_bookmark10) are shown in [Table 5.3](#_bookmark171). Furthermore, comparing the performance of Mixtral and Claude Opus with the newly obtained [EX](#_bookmark10), Claude Opus significantly out- performs Mixtral, with a p-value of 4*.*30 10*−*23, concluding that the Claude Opus model is the better of the two models.

·

Table 5.3: [EX](#_bookmark10) of Mixtral [FT](#_bookmark12) 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 [E](#_bookmark10)X ↑ Mixtral [FT](#_bookmark12) + 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.

## Fine-Tuned Mixtral with Beam Search

Using the [BIRD](#_bookmark2) evaluation script, both Mixtral [FT](#_bookmark12) 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](#_bookmark170).

Perhaps fine-tuning the model on other higher quality Text-to-[SQL](#_bookmark29) data than the [BIRD](#_bookmark2) training dataset would have improved the model’s Text-to-[SQL](#_bookmark29) abil- ity. This is supported in [[95](#_bookmark294)], who show that the quality of data is very impor- tant when training [LLMs](#_bookmark17).

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.

## 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](#_bookmark2) 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](#_bookmark17) perform significantly better on tasks from popular Github repositories that existed before the training cutoff [[101](#_bookmark300)], 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](#_bookmark178), 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.

How many male customers who are living

#### Utterance

in North Bohemia have average salary greater than 8000?

Male means that gender = ’M’; A3 refers to region; A11 pertains to average salary.

**Evidence**

SELECT COUNT(\*) AS count

FROM client c JOIN district d

#### Predicted query

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 For- mula 1 database, which contains data mainly regarding Formula 1 seasons and races throughout the entire history of Formula 1. For example, [Table 6.2](#_bookmark180) shows that the model has an understanding of Formula 1 races naming conventions,

although it is not explicitly mentioned in the question.[∗](#_bookmark182) Mixtral [FT](#_bookmark12) 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.

What time did the the 2010’s Formula\_1 race took place on the Abu Dhabi Circuit?

#### Evidence -

**Utterance**

SELECT T2.date, T2.time FROM circuits AS T1 INNER JOIN races AS T2

#### Gold query

**Predicted query**

ON T2.circuitID = T1.circuitId

WHERE T2.year = 2010

AND T2.name = ’Abu Dhabi Grand Prix’

SELECT time FROM races WHERE year = 2010 AND

circuitId = (

SELECT circuitId FROM circuits

WHERE name = ’Abu Dhabi Grand Prix’

);

## The Error Analysis

Around 40 hours was spent studying the evaluation data classified as incorrect by the evaluation of Mixtral [FT](#_bookmark12) 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](#_bookmark29).

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](#_bookmark10) 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 con- sidering 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.

## 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](#_bookmark2) evalua- tion dataset [[102](#_bookmark301)], 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](#_bookmark185), 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](#_bookmark193). Example 1 shows incorrect us- age 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 some- times 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](#_bookmark29) 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](#_bookmark186). 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](#_bookmark194), 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 el- ements 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

#### MOLECULE

moleculeId

**ATOM**

**BOND**

**CONNECTED**

bondId

atomId2

atomId

bondType

moleculeId

bondId

element

moleculeId

atomId

label

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, indi- cating 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](#_bookmark202)] 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](#_bookmark29) tool. However, such language could negatively impact the [FT](#_bookmark12) process, which could be one of the reasons why Mixtral has a large proportion of misunderstandings in the error analysis.

### Evaluation

The evaluation script available by [BIRD](#_bookmark2) misses some correct classifications as evident by [Section 5.2](#_bookmark169). However, there is no simple solution to automat- ing evaluation such that the script can detect all semantically correct samples without also misclassifying incorrect samples as correct without manual eval- uation. For example, if the gold query returns string Y and the model returns YES, then the result should be considered semantically correct. Another ex- ample 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 utter- ance.

## Ethical Considerations

There are some dangers in providing Text-to-[SQL](#_bookmark29) as a service. For exam- ple, 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](#_bookmark29) 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](#_bookmark29) 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](#_bookmark29). However, it will make mistakes, often ones which are not easily de- tectable by only studying the output when executing the [SQL](#_bookmark29). Hence, it must be used with care and the users of such a system should have access to the [SQL](#_bookmark29) queries to be able to validate that the generated query is correct.

The [BIRD](#_bookmark2) dataset uses open and accessible databases, and the databases do not contain any confidential information. All information in the databases in [BIRD](#_bookmark2) is public and accessible [[3](#_bookmark202)].

## Environmental Impacts

The sustainability aspect should also be considered. Training [LLMs](#_bookmark17) can use a lot of electricity. The training of GPT-3 emitted around 500 tonnes of CO2eq and consumed over 1 GWh of electricity [[58](#_bookmark257)]. It is therefore important to keep in mind the environmental impact of training and testing [LLMs](#_bookmark17). However, [FT](#_bookmark12) and inference consume a lot less electricity compared to the pre-training.

The A100 [GPU](#_bookmark15) used in this study consumes 300W at maximum usage, and the [FT](#_bookmark12) process of Mixtral took roughly 3 hours. The evaluation process took roughly 5 hours, but with lower energy consumption as the [GPU](#_bookmark15) was not fully utilized. The evaluation process was run three times, once before [FT](#_bookmark12) and once with and without beam search. Hence, a rough estimate for the electricity consumption is 18h 300W = 5.4kWh. Converting the consumed energy into CO2eq accurately is nontrivial as there does not exist any public information regarding the [Power Usage Effectiveness (PUE)](#_bookmark22) 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](#_bookmark22) used by the studied pre- trained models in [[58](#_bookmark257)] is used. The carbon intensity of the energy grid is in the range of 57-429gCO2eq/kWh and the data center [PUE](#_bookmark22) is in the range 1.08 to 1.2. Using these numbers, the estimated carbon emissions from the Mixtral [FT](#_bookmark12) and evaluation process are in the range of 332g-2 780g of CO2eq.

×

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**

**Evidence**

**Gold query**

**Explanation**

List the players’ api id who had the highest above average overall ratings in 2010.

highest above average overall ratings refers to MAX(overall\_rating); in 2010 refers to substr(date,1,4)

= ’2010’;

SELECT

player\_api\_id FROM

Player\_Attributes WHERE SUBSTR(‘date‘, 1,

4) = ’2010’ ORDER BY

overall\_rating DESC LIMIT 1

Multiple players has the same highest rating, should list all of them instead of using LIMIT 1.

List the driver’s ID of the top five driver, by descending order, the fastest time during the first lap of the race.

fastest time refers to Min(time);

SELECT driverId FROM lapTimes WHERE lap = 1 ORDER BY time LIMIT 5

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?

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

#### Evidence

**Gold query**

= ’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

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 accura- cies of 82 % and 67 %, respectively, when removing incorrect data points and adding semantically correct data points missed by the evaluation script. The [BIRD](#_bookmark2) evaluation dataset contains roughly 17 % gold [SQL](#_bookmark29) 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 impor- tant, 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](#_bookmark29) generation could be a useful tool for programmers in creating [SQL](#_bookmark29) queries.

The main contribution of the thesis highlight that the data quality of the entire [BIRD](#_bookmark2) evaluation dataset contains 17 % gold [SQL](#_bookmark29) 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](#_bookmark301)].

## Future Work

In this thesis, both fine-tuning Mixtral on [BIRD](#_bookmark2) to improve its Text-to-[SQL](#_bookmark29) performance and using pre-trained Claude Opus have been evaluated on the [BIRD](#_bookmark2) evaluation dataset. In future work, it would be beneficial to fine-tune models on other Text-to-[SQL](#_bookmark29) datasets in order to explore fine-tuning further,

as high quality data is of importance in the fine-tuning process. Further- more, using more advanced prompting techniques for Claude Opus could in- crease performance, for example by providing example column values to the model and descriptions of the meaning of each column. Furthermore, [Few-](#_bookmark11) [Shot](#_bookmark11) 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](#_bookmark33), [7](#_bookmark43), and [8](#_bookmark45).]
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](#_bookmark33) and [5](#_bookmark40).]

1. 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](#_bookmark34), [28](#_bookmark101), [29](#_bookmark104), [30](#_bookmark108), [60](#_bookmark187), and [61](#_bookmark189).]

1. 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](#_bookmark34), [12](#_bookmark58), [25](#_bookmark93), [34](#_bookmark119), [35](#_bookmark120), [36](#_bookmark122), [37](#_bookmark126), [51](#_bookmark164),

and [52](#_bookmark167).]

1. 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](#_bookmark34), [25](#_bookmark93), [34](#_bookmark119), [35](#_bookmark120), [36](#_bookmark122), [37](#_bookmark126),

[51](#_bookmark164), and [52](#_bookmark167).]

1. 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](#_bookmark34), [13](#_bookmark61), [18](#_bookmark74), and [22](#_bookmark86).]

1. 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](#_bookmark34), [12](#_bookmark58), [25](#_bookmark93), [33](#_bookmark117), [36](#_bookmark122), [37](#_bookmark126), [51](#_bookmark164), and [52](#_bookmark167).]
2. 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](#_bookmark34).]
3. 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](#_bookmark34).]
4. A. Pavlo and M. Aslett, “What’s really new with newsql?” *ACM Sigmod Record*, vol. 45, no. 2, pp. 45–55, 2016. [Page [5](#_bookmark40).]
5. 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](#_bookmark43) and [8](#_bookmark45).]

1. 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](#_bookmark43).]

1. 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](#_bookmark45).]

1. 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](#_bookmark45) and [11](#_bookmark54).]
2. 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](#_bookmark45), [11](#_bookmark54), [29](#_bookmark104), and [30](#_bookmark108).]
3. 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](#_bookmark45) and [11](#_bookmark54).]

1. 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](#_bookmark49), [10](#_bookmark52), and [11](#_bookmark54).]

1. M.-C. De Marneffe and J. Nivre, “Dependency grammar,” *Annual Review of Linguistics*, vol. 5, pp. 197–218, 2019. [Page [10](#_bookmark52).]
2. 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](#_bookmark52).]
3. P. Shaw, J. Uszkoreit, and A. Vaswani, “Self-attention with relative position representations,” *arXiv preprint arXiv:1803.02155*, 2018. [Page [10](#_bookmark52).]
4. 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](#_bookmark52).]
5. 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](#_bookmark52).]
6. 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](#_bookmark52), [12](#_bookmark58), [34](#_bookmark119), and [37](#_bookmark126).]
7. 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](#_bookmark54) and [12](#_bookmark58).]
8. 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](#_bookmark54).]
9. 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](#_bookmark54) and [12](#_bookmark58).]
10. 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](#_bookmark58).]

1. 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](#_bookmark58).]
2. N. De Cao, G. Izacard, S. Riedel, and F. Petroni, “Autoregressive entity retrieval,” *arXiv preprint arXiv:2010.00904*, 2020. [Page [12](#_bookmark58).]
3. 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](#_bookmark58).]
4. 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](#_bookmark58).]
5. 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](#_bookmark58).]

1. 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](#_bookmark58) and [25](#_bookmark93).]

1. 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](#_bookmark58), [25](#_bookmark93), [35](#_bookmark120), and [37](#_bookmark126).]

1. X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowd- hery, and D. Zhou, “Self-consistency improves chain of thought reasoning in language models,” *arXiv preprint arXiv:2203.11171*, 2022. [Page [13](#_bookmark61).]
2. 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](#_bookmark61).]

1. 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](#_bookmark61), [25](#_bookmark93), [35](#_bookmark120), [36](#_bookmark122), and [37](#_bookmark126).]
2. 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](#_bookmark61), [33](#_bookmark117), [36](#_bookmark122), and [37](#_bookmark126).]
3. 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](#_bookmark61).]

1. 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](#_bookmark61) and [15](#_bookmark66).]

1. A. Radford, K. Narasimhan, T. Salimans, I. Sutskever *et al.*, “Improving language understanding by generative pre-training,” 2018. [Pages [13](#_bookmark61), [15](#_bookmark66), and [18](#_bookmark74).]
2. 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](#_bookmark61) and [15](#_bookmark66).]
3. 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](#_bookmark63).]
4. 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](#_bookmark63).]

1. 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](#_bookmark66).]
2. P. Gage, “A new algorithm for data compression,” *The C Users Journal*, vol. 12, no. 2, pp. 23–38, 1994. [Page [15](#_bookmark66).]
3. 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](#_bookmark66).]

1. 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](#_bookmark66).]

1. T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, “Distributed representations of words and phrases and their composi- tionality,” *Advances in neural information processing systems*, vol. 26, 2013. [Page [15](#_bookmark66).]
2. 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](#_bookmark66).]
3. 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](#_bookmark69), [17](#_bookmark71), [18](#_bookmark74), [20](#_bookmark79), [21](#_bookmark84), and [41](#_bookmark135).]
4. 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](#_bookmark69).]
5. J. Zheng, S. Ramasinghe, and S. Lucey, “Rethinking positional encoding,” *arXiv preprint arXiv:2107.02561*, 2021. [Page [16](#_bookmark69).]
6. J. Su, M. Ahmed, Y. Lu, S. Pan, W. Bo, and Y. Liu, “Roformer: En- hanced transformer with rotary position embedding,” *Neurocomputing*, vol. 568, p. 127063, 2024. [Pages [16](#_bookmark69) and [41](#_bookmark135).]
7. 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](#_bookmark69).]
8. 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](#_bookmark69).]
9. 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](#_bookmark71).]
10. 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](#_bookmark86)

and [62](#_bookmark192).]

1. 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](#_bookmark86).]

1. 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](#_bookmark86).]

1. 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](#_bookmark86), [23](#_bookmark89), and [24](#_bookmark91).]
2. 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](#_bookmark89).]

1. 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](#_bookmark89).]
2. 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](#_bookmark89).]
3. 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](#_bookmark89).]
4. T. Dettmers, A. Pagnoni, A. Holtzman, and L. Zettlemoyer, “Qlora: Eﬀicient finetuning of quantized llms,” *arXiv preprint arXiv:2305.14314*, 2023. [Pages [24](#_bookmark91), [41](#_bookmark135), and [46](#_bookmark149).]
5. O. Rubin, J. Herzig, and J. Berant, “Learning to retrieve prompts for in- context learning,” *arXiv preprint arXiv:2112.08633*, 2021. [Page [25](#_bookmark93).]
6. 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](#_bookmark93).]
7. W. Fedus, J. Dean, and B. Zoph, “A review of sparse expert models in deep learning,” *arXiv preprint arXiv:2209.01667*, 2022. [Pages [25](#_bookmark93) and [26](#_bookmark95).]
8. 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](#_bookmark95).]
9. 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](#_bookmark95).]
10. 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](#_bookmark95).]
11. 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](#_bookmark95) and [42](#_bookmark138).]

1. A. Eliseev and D. Mazur, “Fast inference of mixture-of-experts language models with offloading,” *arXiv preprint arXiv:2312.17238*, 2023. [Page [26](#_bookmark95).]
2. 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](#_bookmark98), [29](#_bookmark104), and [30](#_bookmark108).]

1. 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](#_bookmark98).]

1. 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](#_bookmark98).]
2. 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](#_bookmark104).]
3. 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](#_bookmark104).]

1. C.-H. Lee, O. Polozov, and M. Richardson, “Kaggledbqa: Realistic evaluation of text-to-sql parsers,” *arXiv preprint arXiv:2106.11455*, 2021. [Page [29](#_bookmark104).]
2. 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](#_bookmark104).]

1. 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](#_bookmark104).]
2. 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](#_bookmark104).]
3. 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](#_bookmark104).]
4. 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](#_bookmark104).]
5. 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](#_bookmark117) and [37](#_bookmark126).]
6. 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](#_bookmark117).]
7. 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](#_bookmark120) and [36](#_bookmark122).]

1. 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](#_bookmark132).]

1. 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](#_bookmark132).]

1. 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](#_bookmark135).]

1. 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](#_bookmark135).]

1. N. Shazeer, “Glu variants improve transformer,” *arXiv preprint arXiv:2002.05202*, 2020. [Page [41](#_bookmark135).]
2. S. Mukherjee, A. Mitra, G. Jawahar, S. Agarwal, H. Palangi, and
3. Awadallah, “Orca: Progressive learning from complex explanation traces of gpt-4,” *arXiv preprint arXiv:2306.02707*, 2023. [Page [41](#_bookmark135).]
4. 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](#_bookmark135) and [55](#_bookmark174).]

1. 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](#_bookmark135).]

1. T. Dettmers, M. Lewis, S. Shleifer, and L. Zettlemoyer, “8-bit opti- mizers via block-wise quantization,” *arXiv preprint arXiv:2110.02861*, 2021. [Page [46](#_bookmark149).]
2. 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](#_bookmark157).]
3. 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](#_bookmark157).]

1. 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](#_bookmark157).]

1. 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](#_bookmark177).]
2. 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](#_bookmark183) and [65](#_bookmark196).]

# Appendix A

**Prompts used for LLMs**

In this Appendix, the prompts for Claude Opus, Mixtral [FT](#_bookmark12) and Pre-Trained Mixtral is shown.

## 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:

## 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