Text to SQL Conversion

Project Report

Application Area

Natural Language Processing
https://github.com/ankdeshm/Text-to-SQL

By

Team 3
Ankita Arvind Deshmukh: 016029585
Sanika Vijaykumar Karwa: 016688815
Priyanka Birju Shah: 016712345

To

Professor Kaikai Liu

On

December 14, 2023

1. Introduction

Text-to-SQL is a task that automatically converts a user's natural language query into SQL. SQL is the query language that is used with relational databases. Given that relational databases house the great bulk of the data in our lives, a solution to this problem could be extremely beneficial. Relational databases seem to be the best choice for businesses such as healthcare, finance, sales, etc. who can't afford to lose transactions and are highly useful to store, manage and manipulate structured data. In the era of data-driven decision-making, the ability to query databases efficiently and accurately has become increasingly crucial. However, the technical barrier of writing SQL queries limits this capability to those with specific programming skills. This project aims to bridge this gap by developing a system capable of translating natural language questions into SQL queries. The significance of this project lies in its potential to democratize data access, enabling users with no formal training in SQL to extract meaningful insights from databases. In this project, we aim to identify various existing approaches for Text-to-SQL conversion, fine-tune them and compare those models to obtain better-performing models.

2. Objective

The primary objective of this project is to identify and finetune various existing approaches for Text-to-SQL conversion which accurately convert user-input natural language questions into corresponding SQL queries. This involves understanding the nuances of natural language, recognizing intent and context, and translating these into structured query language format. The project aims to make this translation as seamless and accurate as possible, catering to a wide range of queries from simple to complex. In order to determine which models work better, we intend to compare the performance of all the existing approaches.

3. Datasets

3.1 WikiSQL

WikiSQL is a large crowd-sourced dataset for developing natural language interfaces for relational databases. It contains 80654 hand-annotated examples of questions and SQL queries distributed across 24241 tables from Wikipedia [1].

3.2 Spider

Spider is a large-scale complex and cross-domain semantic parsing and text-to-SQL dataset annotated by 11 Yale students The goal of the Spider challenge is to develop natural language interfaces to cross-domain databases. Each instance in Spider represents a natural language question and the equivalent SQL query. Spider contains train: 7000 questions and SQL query pairs of training data 1034 questions and SQL query pairs of dev data [7].

3.3 sql-create-context

This dataset builds from WikiSQL and Spider. There are 78,577 examples of natural language queries, SQL CREATE TABLE statements, and SQL Query answering the question using the CREATE statement as context. This dataset was built with text-to-sql LLMs in mind, intending to prevent hallucination of column and table names often seen when trained on text-to-sql datasets. The CREATE TABLE statement can often be copy and pasted from different DBMS and provides table names, column names and their data types [8].

4. Methodology

4.1 Seq2SQL

Seq2SQL is a deep neural network model to obtain SQL queries from corresponding natural language queries. This is done using Reinforcement Learning. This is the baseline model for the task of Text-to-SQL. It uses rewards from in-the-loop query execution on top of the database to further learn a policy to generate the query. It leverages the structure of SQL to prune the space of generated queries which simplifies the problem of generation. It takes questions and columns of the table as input. It generates the SQL query which during training is performed against the database. The result obtained from execution is the reward to train the reinforcement learning algorithm [1].

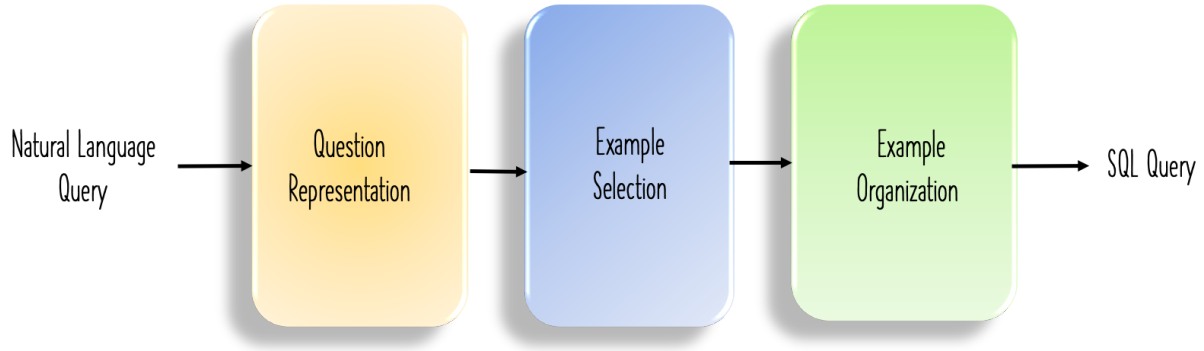


Fig. 1. Seq2SQL Architecture

The model utilizes the attentional sequence to sequence neural semantic parser and further we have the classifier to predict the aggregation operation of the query. Then, there is the augmented pointer network to generate the query conditions token-by-token from an input sequence, and lastly, the policy-based reinforcement learning algorithm to handle the generation of unordered query conditions [1]. The original code to reproduce the results for Seq2SQL is taken from [3] and it is modified to resolve dependency errors.

4.2 SQLNet

SQLNet solves the problem of Text-to-SQL by avoiding the sequence-to-sequence structure and uses a sketch-based approach so that the prediction can be done by only taking the previous considerations into account. This is an extension of the Seq2SQL model without reinforcement learning as they found that improvement from the reinforcement learning model is limited and avoids the "order matters" problem. It uses a novel attention structure called column attention to boost performance [2].

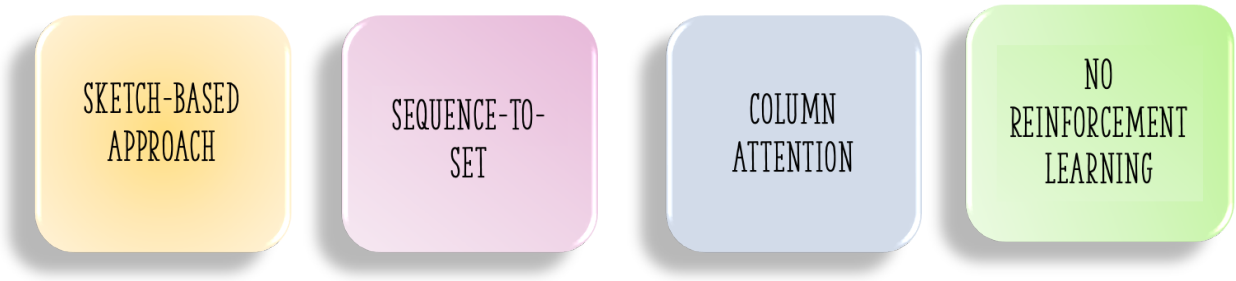


Fig. 2. SQLNet Architecture

This model employs the sketch-based approach which aligns with the SQL grammar. It just needs to fill in the slots in the sketch rather than predicting the output grammar and the content. Then, it employs the sequence-to-set model to generate SQL queries where order doesn't matter. Further, there is a column attention mechanism which boosts the model's performance. And lastly, it eliminates the need for reinforcement learning and improves performance by ~4% [2]. The original code to reproduce the results for SQLNet is taken from [3] and it is modified to resolve dependency errors.

4.3 C3-SQL

C3-SQL model is a ChatGPT-based zero-shot Text-to-SQL method. It is tested on the Spider dataset and is the state-of-the-art zero-shot Text-to-SQL method. Zero-shot learning means that the model can learn and understand the task the first time it is asked without giving any prior examples or fine-tuning it. C3 consists of three key components, Clear Prompting(CP),

Calibration with Hints(CH) and Consistent Output(CO) which corresponds to the model input, model bias and model output respectively. Because this is a zero-shot method, it uses approximately 1000 tokens per query. Clear Prompting corresponds to clean and effective input with precise instructions given to the model with two main parts namely clear layout and clear context. Calibration with Hints provides some hints for the model biases to be resolved. Consistent Output overcomes the randomness and uncertainty of the output using a self consistency method which is finding the unique right answer for multiple different paths, which would improve the reliability of the outputs produced. Overall, C3SQL provides a systematic approach for GPT-based Text-to-SQL search from the perspective of model input, bias and output [4].

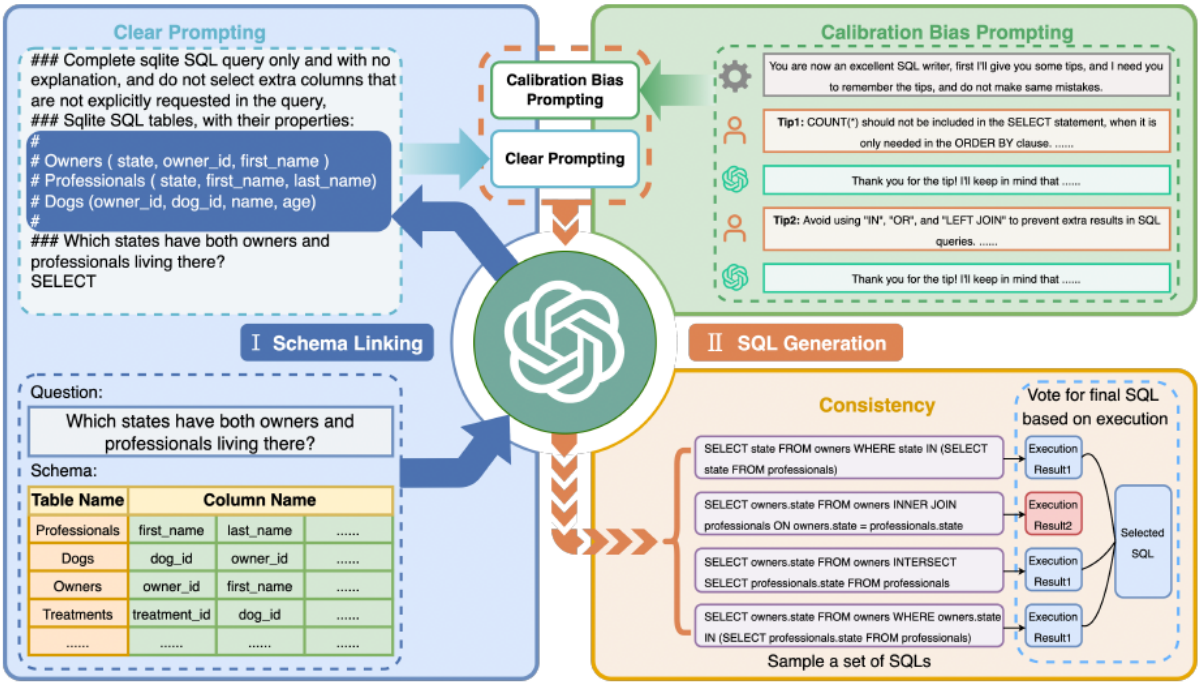


Fig. 3. C3-SQL overview [4]

4.4 DIN-SQL

The paper "DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction" [5] presents a technique to enhance the capacity of Large Language Models (LLMs) to produce SQL queries from natural language. With this method, difficult Text-to-SQL jobs are broken down into easier subtasks, and the final query is built using the solutions to those subtasks. This method demonstrated state-of-the-art performance on the Spider dataset when evaluated with GPT-4. It consists of four modules that improve the robustness and accuracy of the created SQL queries: self-correction, SQL generation, query categorization and decomposition, and schema linkage as shown in Fig. 4 [5].

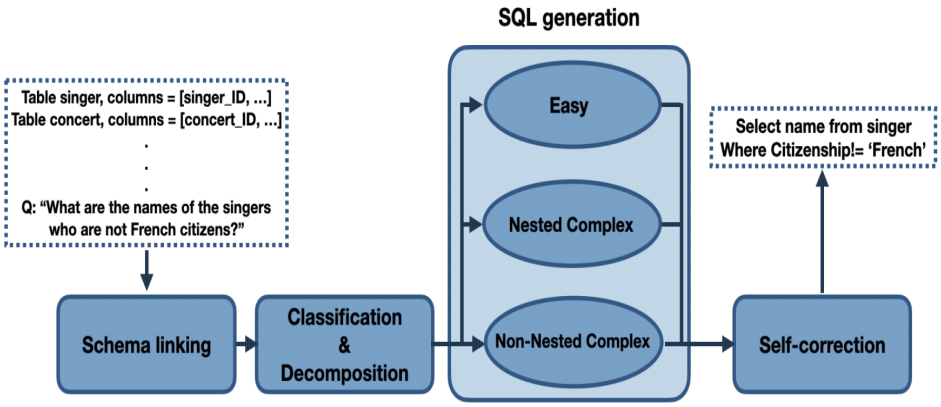


Fig. 4. DIN-SQL Overview

1. **Schema Linking:** The first module in the DIN-SQL model shown in Fig. 5. , schema linking, is essential for connecting certain sections of a database schema to natural language queries. It significantly simplifies creating complex queries and adjusting to various domains. The Large Language Model's (LLM) highest error rate is seen in this module. Ten arbitrary samples from the Spider dataset are used in a prompt-based, chain-of-thought technique for schema linking. In addition to collecting potential entities and values from the query, this method includes identifying column names and matching tables.

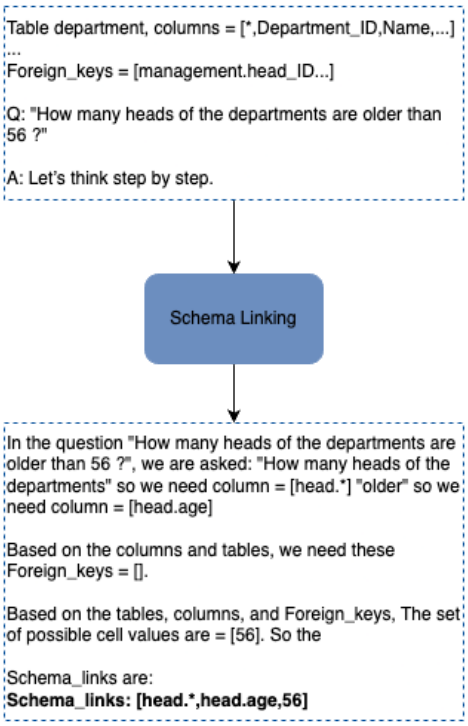


Fig. 5. DIN-SQL Schema Linking Module

2. **Classification and Decomposition:** Complex queries and join operations are addressed in DIN-SQL's second module, Classification & Decomposition. It divides queries into three categories: non-nested complex (needs joins but no sub-queries), nested complex (needs joins, sub-queries, and set operations), and easy (single-table, no joins or nesting). Additionally, the module finds sub-queries for nested queries and determines which tables need to be combined. Each query class has a different set of prompts, which helps with precise query formulation. The paper provides an example of the input and output of the module.

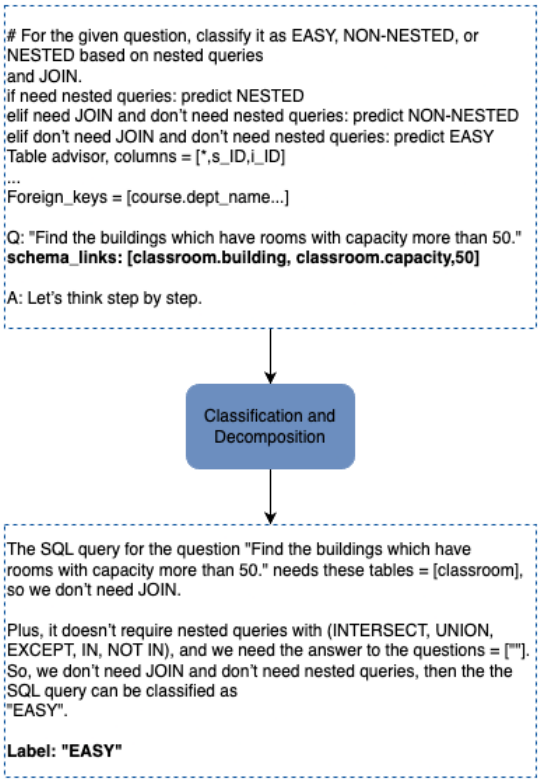


Fig. 6. DIN-SQL Classification and Decomposition Module

3. **SQL Generation:** DIN-SQL's SQL Generation Module handles the challenge of translating natural language queries into SQL. For the three query classes—easy, non-nested complex, and nested complex—it employs several strategies. It uses simple prompting for simple requests. In non-nested complicated queries, join operations are handled using an intermediate representation such as NatSQL. The most complex inquiries are nested complex queries, which involve solving sub-queries prior to the final SQL generation. To close the gap between natural language and SQL, each class makes use of particular formats and prompts.

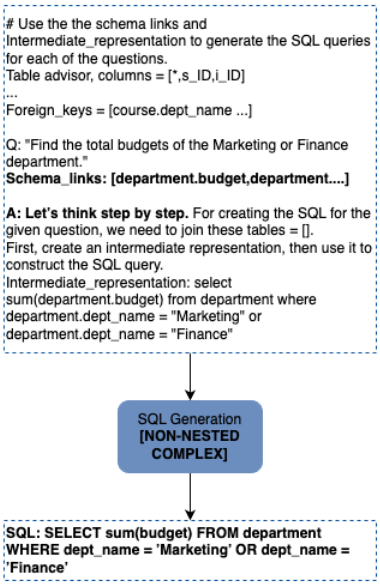


Fig. 7. DIN-SQL SQL Generation Module

4. **Self-Correction:** Lastly, the self-correction module is designed to fix little mistakes in SQL queries that are produced by LLMs, such as duplicate or missing terms. Errors are corrected by the model in a zero-shot scenario, which eliminates the need for further examples. The model is asked to find and fix faults using two different kinds of prompts: a generic question that asks the model directly, and a soft prompt that gently advises looking for possible problems without assuming the query is flawed.

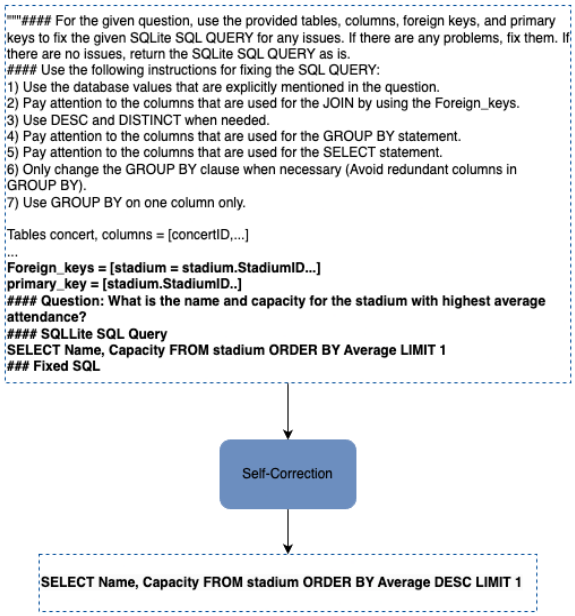


Fig. 8. DIN-SQL Self-Correction Module

4.5 DAIL-SQL

Currently, DAIL-SQL stands as one of the most advanced methods in the field of translating text to SQL. It thoroughly evaluates and tests various existing techniques of prompt engineering and determines the most effective method for question representation, as well as the selection and arrangement of examples. The approach to representing questions is zero-shot, whereas the strategies for selecting and organizing examples are few-shot. For the representation of questions, it identifies examples by considering information from both the question and its corresponding query, arranging them in a way that maintains the association between the question and its SQL translation. In the process of selecting examples, the method involves picking the best *k* examples, showcasing their questions and queries within a Code Representation Prompt. This prompt is then input into a large-scale language model to forecast the SQL query. Subsequently, the Jaccard similarity index is employed to evaluate the resemblance between the example candidates and the predicted query based on their query similarity. Furthermore, DAIL-SQL introduces a refined solution to the issue of token efficiency, a common challenge in other large language model-based approaches [6].

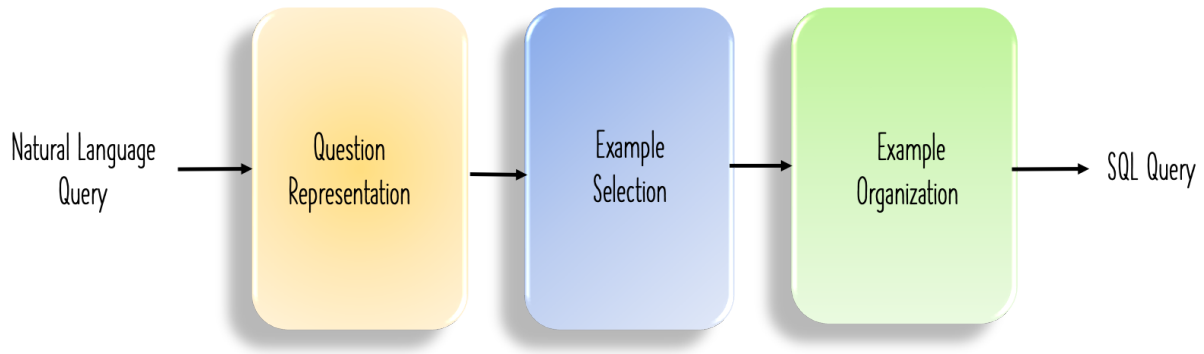


Fig. 9. DAIL-SQL Overview

4.6 Transformer-based models from Hugging Face

4.6.1 gpt2Medium_text_to_sql

The model "gpt2Medium_text_to_sql" on Hugging Face is a fine-tuned version of GPT-2 Medium, designed to translate English questions into SQL queries. The user provides a natural language query, which the model converts into SQL using a pre-trained tokenizer and model. The model is evaluated on its ability to generate accurate SQL queries from given inputs, with training hyperparameters and loss metrics provided to gauge its effectiveness. It has been trained using the dataset sql-create-context [8]. The model was trained with the following

hyperparameters: it underwent 1 training epoch, using a batch size of 2 for each device. Gradient accumulation, which allows for training with larger batch sizes than memory may otherwise permit, was set to accumulate over 9 steps. The learning rate was set at 5e-5, and weight decay, which is a regularization technique to prevent overfitting, was applied at a rate of 0.01.

4.6.2 t5-small-awesome-text-to-sql

The "t5-small-awesome-text-to-sql" model is a light-weight solution built on the T5-small architecture, intended for converting natural language into SQL queries. The T5, or Text-to-Text Transfer Transformer, is a deep learning model designed to handle a variety of NLP tasks by framing them as a text-to-text problem. It uses a unified text-based framework where tasks such as translation, question-answering, and classification are all formatted in the same way: taking text as input and producing new text as output. This approach allows for flexibility and simplicity in handling diverse tasks. The "small" variant of T5 offers a more lightweight version, maintaining a balance between performance and computational efficiency, ideal for tasks where resource constraints are a consideration. The model “t5-small-awesome-text-to-sql” is equipped to handle complex queries involving multiple tables and joins. The model was trained using a combination of the sql-create-context [8] and Text-to-sql-v1 datasets available on Hugging Face, making it versatile for various analytical applications. It is originally implemented via PyTorch and HuggingFace tools, running on an A100-80 hardware infrastructure.

4.6.3 Mistral-7B-SQL

The Mistral-7B-SQL is a transformer model specifically fine-tuned for generating SQL queries from natural language instructions. It implements architectural features such as Grouped-Query and Sliding-Window Attention, utilizing a Byte-fallback BPE tokenizer and LoRA to optimize matrix ranks, reducing computational load. The model was trained over 10 epochs on the sql-create-context dataset [8], which contains 78.6k records. Training was conducted on an Nvidia A100 GPU, achieving a low loss, indicative of high learning efficiency. It supports a maximum sequence length of 2048 with an effective batch size of 4, employing mixed precision training (tf32). The model is available on Hugging Face's platform for use in various NLP tasks related to SQL generation.

4.6.4 CodeLlama-7b-Instruct-SQL

The CodeLlama-7b-Instruct-SQL by Machinists is a model fine-tuned for SQL generation from natural language instructions, utilizing the LoRA technique for computational efficiency. It was trained on the sql-create-context dataset [8] with 78.6k records for 5 epochs and utilizes a maximum sequence length of 1024. The model operates on an Nvidia A100 80GB GPU, promising precision with mixed precision training (tf32) and efficient model loading (bf16). It's

adapted from the CodeLlama-7b-Instruct-hf model, designed to handle instruction-based prompts enclosed by specific tokens for fine-tuning. CodeLlama-7b-hf is a collection of pretrained and fine-tuned generative text models from 7 billion parameters.

4.6.5 CodeLlama-13b-Instruct-SQL

The CodeLlama-13b-Instruct-SQL model is a fine-tuned version of CodeLlama-13b-hf designed for generating SQL queries from natural language instructions. CodeLlama-13b-hf is a collection of pretrained and fine-tuned generative text models from 13 billion parameters. It utilizes the LoRA fine tuning technique for computational efficiency, trained for 4 epochs on the sql-create-context dataset [8] containing 78.6k records. It supports a maximum sequence length of 1024 and uses bf16 precision for model loading. Training was conducted on an Nvidia A100 80GB GPU. This model requires prompts to be surrounded by specific tokens to leverage instruction fine-tuning.

4.6.6 squeal

The model "squeal" is designed to generate SQL queries from text. It employs the use of a pre-trained model, "llama-2-7b-guanaco-instruct-sharded," as its base. "squeal" uses advanced techniques for model loading and optimization, including BitsAndBytesConfig for efficient memory usage. This model is particularly suited for applications where SQL queries need to be generated from structured natural language inputs. It was trained using the sql-create-context dataset [8] and is accessible through the Hugging Face platform for various NLP tasks related to SQL generation.

4.6.7 T5-LM-Large-text2sql-spider

The "T5-LM-Large-text2sql-spider" model is designed to generate SQL queries from natural language prompts. It's fine-tuned from the T5-large-LM-adapt checkpoint on the Spider [7] and Spider-Syn datasets. Unique to its training is the inclusion of the database schema in the input questions, enhancing its ability to generate applicable SQL queries. The model’s training used the AdaptationArguments with specific parameters like a learning rate of 5e-5, 10 training epochs, and gradient accumulation steps of 8. The model's approach allows for better generalization across various database schemas.

5. User Interface

This project's user interface (UI) is a Flask web application that converts natural language queries to SQL utilizing a variety of models. Users can enter their query, select from a variety of databases and models, and get the matching SQL conversion as shown in Fig. 10 below. Hugging Face's Transformers library is integrated into the backend for model implementations, and it manages query processing and model selection. The application offers comprehensive model information, including architecture and correctness, and is both interactive and user-friendly. Flask's rendering system handles the front-end design with the help of HTML templates.This user interface can be accessed via http://127.0.0.1:5000/ or on a local server, localhost. This approach ensures data privacy and quick response times, as processing happens on the local machine.

In the UI for the GPT-based models like C3, DIN-SQL, and DAIL-SQL, the application does not dynamically generate SQL queries in real-time. Instead, it displays pre-generated predictions from a predetermined text file. When a user inputs a query, the system searches these files for a corresponding answer rather than using the models to generate a new response on the fly. This approach simplifies the backend processing but limits the output to pre-existing examples covered in the text files.

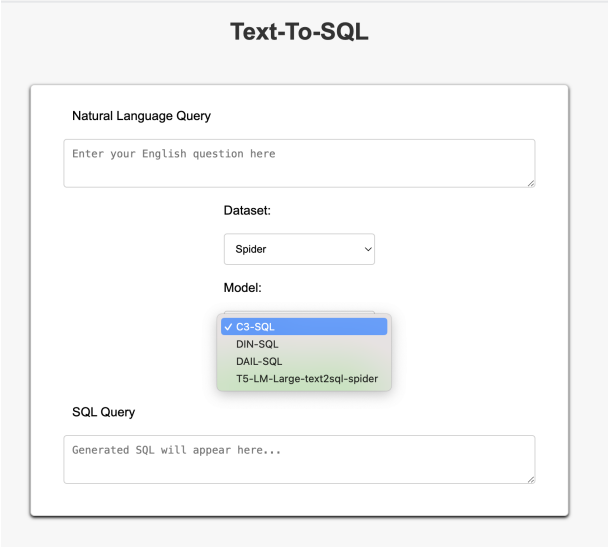


Fig. 10.1. Text-to-SQL UI Web Application

Fig. 10.2. T5-small example query

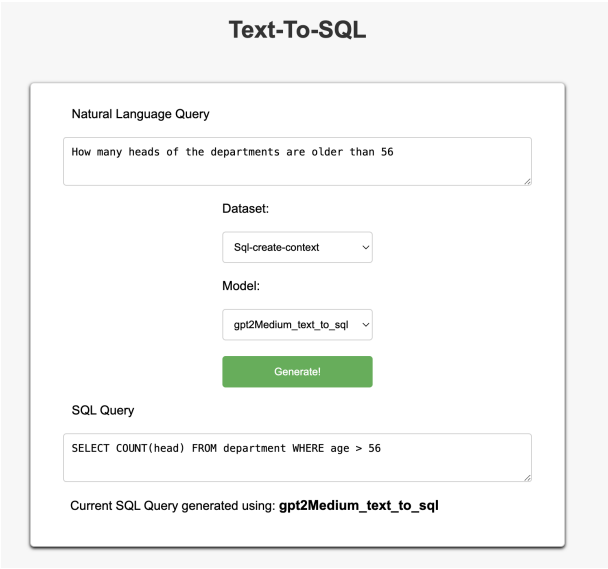


Fig.10.3. gpt2Medium_text_to_sql example query

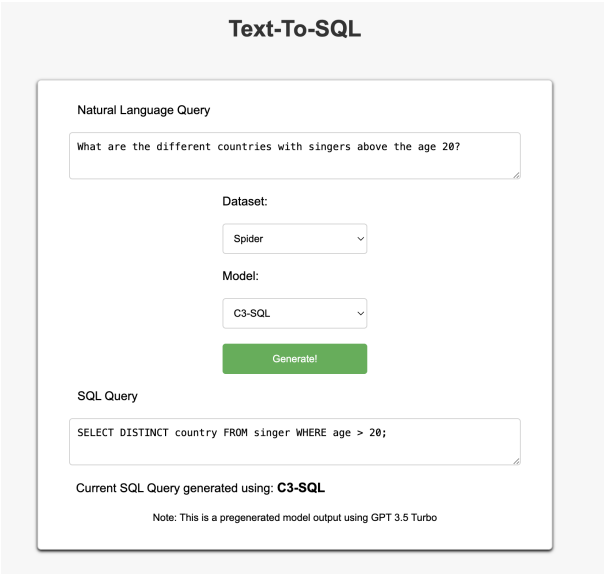


Fig. 10.4. C3-SQL example query

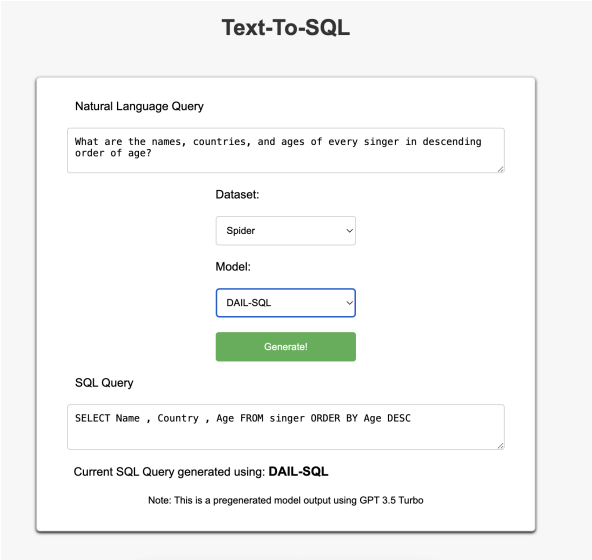


Fig. 10.5. DAIL-SQL example query

Fig. 10.6. DIN-SQL example query

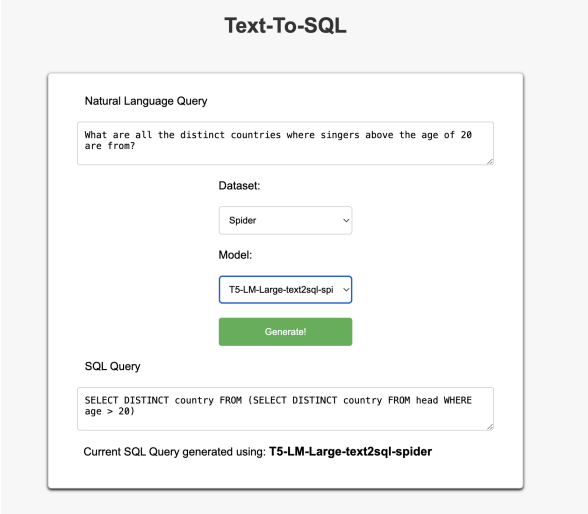


Fig. 10.7. T5-LM-Large-text2sql-spider ex query

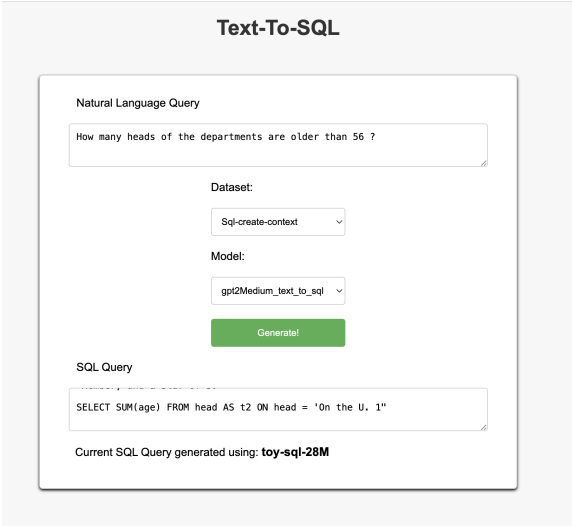


Fig. 10.8. toy-sql-28M example query

6. Results

Evaluation of the Baseline

Seq2SQL

Seq2SQL without Reinforcement Learning

```
Dev acc_qm: breakdown on (agg, sel, where): (0.41345525800130634, array([0.9248857 , 0.85189419, 0.48873285]))
Dev execution acc: 0.4988569562377531
Test acc_qm: breakdown on (agg, sel, where): (0.41623122142390595, array([0.92039517, 0.85336381, 0.49706074]))
Test execution acc: 0.505388634879164
```

Seq2SQL with Reinforcement Learning

```
Dev acc_qm: breakdown on (agg, sel, where): (0.41345525800130634, array([0.9248857 , 0.85189419, 0.48873285]))
Dev execution acc: 0.4988569562377531
Test acc_qm: breakdown on (agg, sel, where): (0.41623122142390595, array([0.92039517, 0.85336381, 0.49706074]))
Test execution acc: 0.505388634879164
```

SQLNet

SQLNet with column attention:

```
Dev acc_qm: breakdown on (agg, sel, where): (0.5183469896686854, array([0.90060563, 0.90963069, 0.60907256]))
Dev execution acc: 0.5949412183826149
Test acc_qm: breakdown on (agg, sel, where): (0.5036528530041567, array([0.90074317, 0.89885376, 0.59566696]))
Test execution acc: 0.5829449552840408
```

SQLNet with column attention & trainable embedding:

```
Dev acc_qm: breakdown on (agg, sel, where): (0.5941099631872699, array([0.90060563, 0.91818074, 0.69540435]))
Dev execution acc: 0.6583541147132169
Test acc_qm: breakdown on (agg, sel, where): (0.5817483310240584, array([0.90074317, 0.91082   , 0.68201285]))
Test execution acc: 0.6476256455472982
```

Evaluation for Hugging Face Models

In the scope of this project, it is important to note that a comprehensive evaluation of the Hugging Face models was not conducted. The primary utilization of these models was restricted to inference purposes only. This decision was influenced by several key factors. Firstly, the Hugging Face models in question were developed based on multiple datasets with the Spider dataset among others. Additionally, constraints related to time and computational memory played a pivotal role in guiding this decision. Therefore, the focus was directed towards leveraging the capabilities of these models for inference only. A thorough evaluation is performed on other models such as Seq2SQL, SQLNet, C3-SQL, DIN-SQL and DAIL-SQL.

Evaluation for the LLM Models:

The generative LLM models trained on the Spider dataset are evaluated using the official Spider evaluation tool provided in [7] and [12]. The models are evaluated based on (1) SQL Difficulty Level, (2) Matching with/without values plugged, (3) Partial and exact matching.

1. **SQL Difficulty Level:**
   First, the following SQL components are defined:
   1. SQL components 1: WHERE, GROUP BY, ORDER BY, LIMIT, JOIN, etc
   2. SQL components 2: EXCEPT, UNION, INTERSECT, NESTED
   3. Others

   Then, the SQL difficulty level is determined by the following criteria:
   a. **Easy**
      If SQL keywords have less than or equal to one component from [SQL components 1] and none from SQL components 2 and Others
   b. **Medium**
      If the SQL satisfies no more than two rules in Others, up to one component from SQL components 1, and none in SQL components 2
   c. **Hard**
      If the SQL satisfies more than two rules in Others, up to two components from SQL components 1, and up to 1 component in SQL components 2
   d. **Extra Hard**
      All the SQLs left

2. **Value String:**
    a. **Not Provided-** The model is expected to generate a value prediction for the value string and is evaluated on the accuracy of the predicted value string as well as the SQL query accuracy.
    b. **Provided-** The model is not expected to generate a value prediction for the value string and instead a list of gold values for each question is provided. This evaluates the accuracy of just the SQL query structure as opposed to the generated value string.
3. **Matching Accuracy:**
    a. **Exact Match Accuracy -** The exact-set match accuracy measures the matched SQL keywords between the ground truth and its corresponding predicted SQL query. This would give less accuracy even if one of the parts of the SQL query is wrong/missing as it measures the exactness of the ground truth.

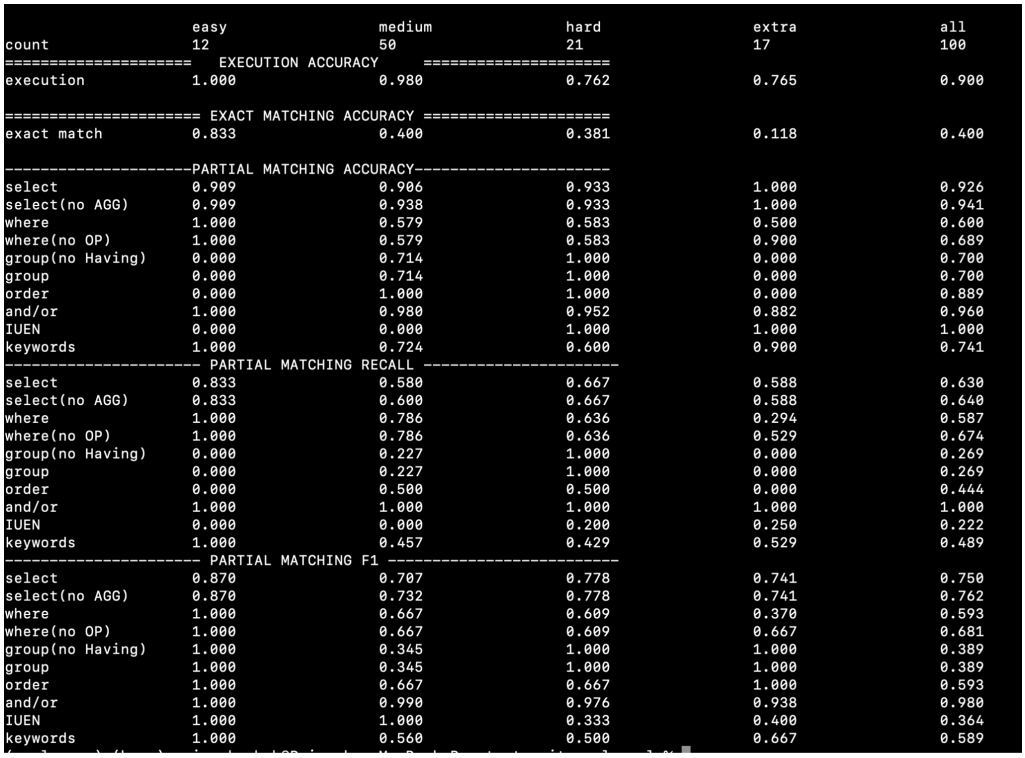
    b. **Execution Accuracy -** The execution accuracy compares the execution output of the predicted SQL query with the ground truth SQL query on some database instances. This is a more precise metric to measure the performance because there could be multiple possible SQL queries for a single natural language question.

1. C3-SQL

This model was tested on 100 examples. The accuracy of the model on Spider dev is 89% and the exact match is 40%. The evaluation accuracy without gold value strings provided is shown in Fig 11.1 and the evaluation accuracy with gold values plugged in is shown in Fig 11.2.

	easy	medium	hard	extra	all
count	12	50	21	17	100
========== EXECUTION ACCURACY ==========					
execution	1.000	0.980	0.714	0.765	0.890
========== EXACT MATCHING ACCURACY ==========					
exact match	0.833	0.400	0.381	0.118	0.400
---------- PARTIAL MATCHING ACCURACY ----------					
select	0.909	0.906	0.933	1.000	0.926
select(no AGG)	0.909	0.938	0.933	1.000	0.941
where	1.000	0.579	0.583	0.500	0.600
where(no OP)	1.000	0.579	0.583	0.900	0.689
group(no Having)	0.000	0.714	1.000	0.000	0.700
group	0.000	0.714	1.000	0.000	0.700
order	0.000	1.000	1.000	0.000	0.889
and/or	1.000	0.980	0.952	0.882	0.960
IUEN	0.000	0.000	1.000	1.000	1.000
keywords	1.000	0.724	0.600	0.900	0.741
---------- PARTIAL MATCHING RECALL ----------					
select	0.833	0.580	0.667	0.588	0.630
select(no AGG)	0.833	0.600	0.667	0.588	0.640
where	1.000	0.786	0.636	0.294	0.587
where(no OP)	1.000	0.786	0.636	0.529	0.674
group(no Having)	0.000	0.227	1.000	0.000	0.269
group	0.000	0.227	1.000	0.000	0.269
order	0.000	0.500	0.500	0.000	0.444
and/or	1.000	1.000	1.000	1.000	1.000
IUEN	0.000	0.000	0.200	0.250	0.222
keywords	1.000	0.457	0.429	0.529	0.489
---------- PARTIAL MATCHING F1 ----------					
select	0.870	0.707	0.778	0.741	0.750
select(no AGG)	0.870	0.732	0.778	0.741	0.762
where	1.000	0.667	0.609	0.370	0.593
where(no OP)	1.000	0.667	0.609	0.667	0.681
group(no Having)	1.000	0.345	1.000	1.000	0.389
group	1.000	0.345	1.000	1.000	0.389
order	1.000	0.667	0.667	1.000	0.593
and/or	1.000	0.990	0.976	0.938	0.980
IUEN	1.000	1.000	0.333	0.400	0.364
keywords	1.000	0.560	0.500	0.667	0.589

Fig 11.1: C3-SQL without plugging



	easy	medium	hard	extra	all
count	12	50	21	17	100
========== EXECUTION ACCURACY ==========					
execution	1.000	0.980	0.762	0.765	0.900
========== EXACT MATCHING ACCURACY ==========					
exact match	0.833	0.400	0.381	0.118	0.400
---------- PARTIAL MATCHING ACCURACY ----------					
select	0.909	0.906	0.933	1.000	0.926
select(no AGG)	0.909	0.938	0.933	1.000	0.941
where	1.000	0.579	0.583	0.500	0.600
where(no OP)	1.000	0.579	0.583	0.900	0.689
group(no Having)	0.000	0.714	1.000	0.000	0.700
group	0.000	0.714	1.000	0.000	0.700
order	0.000	1.000	1.000	0.000	0.889
and/or	1.000	0.980	0.952	0.882	0.960
IUEN	0.000	0.000	1.000	1.000	1.000
keywords	1.000	0.724	0.600	0.900	0.741
---------- PARTIAL MATCHING RECALL ----------					
select	0.833	0.580	0.667	0.588	0.630
select(no AGG)	0.833	0.600	0.667	0.588	0.640
where	1.000	0.786	0.636	0.294	0.587
where(no OP)	1.000	0.786	0.636	0.529	0.674
group(no Having)	0.000	0.227	1.000	0.000	0.269
group	0.000	0.227	1.000	0.000	0.269
order	0.000	0.500	0.500	0.000	0.444
and/or	1.000	1.000	1.000	1.000	1.000
IUEN	0.000	0.000	0.200	0.250	0.222
keywords	1.000	0.457	0.429	0.529	0.489
---------- PARTIAL MATCHING F1 ----------					
select	0.870	0.707	0.778	0.741	0.750
select(no AGG)	0.870	0.732	0.778	0.741	0.762
where	1.000	0.667	0.609	0.370	0.593
where(no OP)	1.000	0.667	0.609	0.667	0.681
group(no Having)	1.000	0.345	1.000	1.000	0.389
group	1.000	0.345	1.000	1.000	0.389
order	1.000	0.667	0.667	1.000	0.593
and/or	1.000	0.990	0.976	0.938	0.980
IUEN	1.000	1.000	0.333	0.400	0.364
keywords	1.000	0.560	0.500	0.667	0.589

Fig 11.2: C3-SQL with plugging

2. DIN-SQL

This model was tested on 63 examples. The accuracy of the model on Spider dev is 93.7% and the exact match is 63.5%. The evaluation accuracy for DIN-SQL without gold value strings provided is shown in Fig 12.1 and the evaluation accuracy with gold values plugged in is shown in Fig 12.2.

	easy	medium	hard	extra	all
count	8	30	15	10	63
==========	EXECUTION ACCURACY	==========			
execution	1.000	0.967	0.800	1.000	0.937
==========	EXACT MATCHING ACCURACY	==========			
exact match	1.000	0.500	0.667	0.700	0.635
----------	PARTIAL MATCHING ACCURACY	----------			
select	1.000	0.741	0.933	1.000	0.867
select(no AGG)	1.000	0.741	0.933	1.000	0.867
where	1.000	0.667	0.692	0.700	0.718
where(no OP)	1.000	0.667	0.692	1.000	0.795
group(no Having)	0.000	0.444	1.000	1.000	0.615
group	0.000	0.444	1.000	1.000	0.615
order	0.000	1.000	1.000	1.000	1.000
and/or	1.000	0.931	1.000	1.000	0.968
IUEN	0.000	0.000	1.000	1.000	1.000
keywords	1.000	0.840	0.733	1.000	0.852
----------	PARTIAL MATCHING RECALL	----------			
select	1.000	0.667	0.933	1.000	0.825
select(no AGG)	1.000	0.667	0.933	1.000	0.825
where	1.000	0.800	1.000	0.700	0.848
where(no OP)	1.000	0.800	1.000	1.000	0.939
group(no Having)	0.000	0.400	1.000	1.000	0.571
group	0.000	0.400	1.000	1.000	0.571
order	0.000	0.625	0.500	1.000	0.643
and/or	1.000	0.964	1.000	1.000	0.984
IUEN	0.000	0.000	0.333	1.000	0.714
keywords	1.000	0.750	0.733	1.000	0.807
----------	PARTIAL MATCHING F1	----------			
select	1.000	0.702	0.933	1.000	0.846
select(no AGG)	1.000	0.702	0.933	1.000	0.846
where	1.000	0.727	0.818	0.700	0.778
where(no OP)	1.000	0.727	0.818	1.000	0.861
group(no Having)	1.000	0.421	1.000	1.000	0.593
group	1.000	0.421	1.000	1.000	0.593
order	1.000	0.769	0.667	1.000	0.783
and/or	1.000	0.947	1.000	1.000	0.976
IUEN	1.000	1.000	0.500	1.000	0.833
keywords	1.000	0.792	0.733	1.000	0.829

Fig 12.1: DIN-SQL without plugging

	easy	medium	hard	extra	all
count	8	30	15	10	63
==========	EXECUTION ACCURACY	==========			
execution	1.000	0.967	0.867	1.000	0.952
==========	EXACT MATCHING ACCURACY	==========			
exact match	1.000	0.500	0.667	0.700	0.635
----------	PARTIAL MATCHING ACCURACY	----------			
select	1.000	0.741	0.933	1.000	0.867
select(no AGG)	1.000	0.741	0.933	1.000	0.867
where	1.000	0.667	0.692	0.700	0.718
where(no OP)	1.000	0.667	0.692	1.000	0.795
group(no Having)	0.000	0.444	1.000	1.000	0.615
group	0.000	0.444	1.000	1.000	0.615
order	0.000	1.000	1.000	1.000	1.000
and/or	1.000	0.931	1.000	1.000	0.968
IUEN	0.000	0.000	1.000	1.000	1.000
keywords	1.000	0.840	0.733	1.000	0.852
----------	PARTIAL MATCHING RECALL	----------			
select	1.000	0.667	0.933	1.000	0.825
select(no AGG)	1.000	0.667	0.933	1.000	0.825
where	1.000	0.800	1.000	0.700	0.848
where(no OP)	1.000	0.800	1.000	1.000	0.939
group(no Having)	0.000	0.400	1.000	1.000	0.571
group	0.000	0.400	1.000	1.000	0.571
order	0.000	0.625	0.500	1.000	0.643
and/or	1.000	0.964	1.000	1.000	0.984
IUEN	0.000	0.000	0.333	1.000	0.714
keywords	1.000	0.750	0.733	1.000	0.807
----------	PARTIAL MATCHING F1	----------			
select	1.000	0.702	0.933	1.000	0.846
select(no AGG)	1.000	0.702	0.933	1.000	0.846
where	1.000	0.727	0.818	0.700	0.778
where(no OP)	1.000	0.727	0.818	1.000	0.861
group(no Having)	1.000	0.421	1.000	1.000	0.593
group	1.000	0.421	1.000	1.000	0.593
order	1.000	0.769	0.667	1.000	0.783
and/or	1.000	0.947	1.000	1.000	0.976
IUEN	1.000	1.000	0.500	1.000	0.833
keywords	1.000	0.792	0.733	1.000	0.829

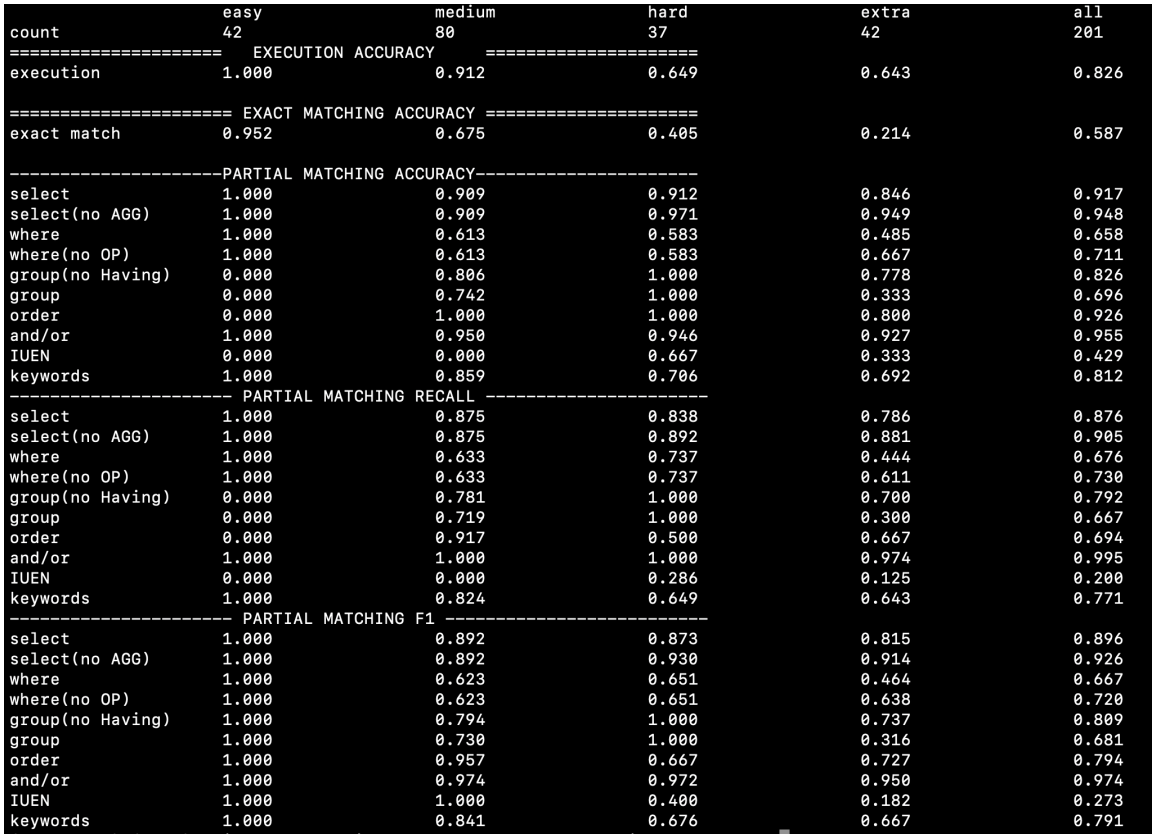
Fig 12.2: DIN-SQL with plugging

3. DAIL-SQL

This model was tested on 200 examples. The accuracy of the model on Spider dev is 76.6% and the exact match is 58.7%. The evaluation accuracy for DAIL-SQL without gold value strings provided is shown in Fig 13.1 and the evaluation accuracy with gold values plugged in is shown in Fig 13.2.

	easy	medium	hard	extra	all
count	42	80	37	42	201
==========	EXECUTION ACCURACY	==========			
execution	0.952	0.887	0.541	0.548	0.766
==========	EXACT MATCHING ACCURACY	==========			
exact match	0.952	0.675	0.405	0.214	0.587
----------	PARTIAL MATCHING ACCURACY	----------			
select	1.000	0.909	0.912	0.846	0.917
select(no AGG)	1.000	0.909	0.971	0.949	0.948
where	1.000	0.613	0.583	0.485	0.658
where(no OP)	1.000	0.613	0.583	0.667	0.711
group(no Having)	0.000	0.806	1.000	0.778	0.826
group	0.000	0.742	1.000	0.333	0.696
order	0.000	1.000	1.000	0.800	0.926
and/or	1.000	0.950	0.946	0.927	0.955
IUEN	0.000	0.000	0.667	0.333	0.429
keywords	1.000	0.859	0.706	0.692	0.812
----------	PARTIAL MATCHING RECALL	----------			
select	1.000	0.875	0.838	0.786	0.876
select(no AGG)	1.000	0.875	0.892	0.881	0.905
where	1.000	0.633	0.737	0.444	0.676
where(no OP)	1.000	0.633	0.737	0.611	0.730
group(no Having)	0.000	0.781	1.000	0.700	0.792
group	0.000	0.719	1.000	0.300	0.667
order	0.000	0.917	0.500	0.667	0.694
and/or	1.000	1.000	1.000	0.974	0.995
IUEN	0.000	0.000	0.286	0.125	0.200
keywords	1.000	0.824	0.649	0.643	0.771
----------	PARTIAL MATCHING F1	----------			
select	1.000	0.892	0.873	0.815	0.896
select(no AGG)	1.000	0.892	0.930	0.914	0.926
where	1.000	0.623	0.651	0.464	0.667
where(no OP)	1.000	0.623	0.651	0.638	0.720
group(no Having)	1.000	0.794	1.000	0.737	0.809
group	1.000	0.730	1.000	0.316	0.681
order	1.000	0.957	0.667	0.727	0.794
and/or	1.000	0.974	0.972	0.950	0.974
IUEN	1.000	1.000	0.400	0.182	0.273
keywords	1.000	0.841	0.676	0.667	0.791

Fig 13.1: DAIL-SQL without plugging



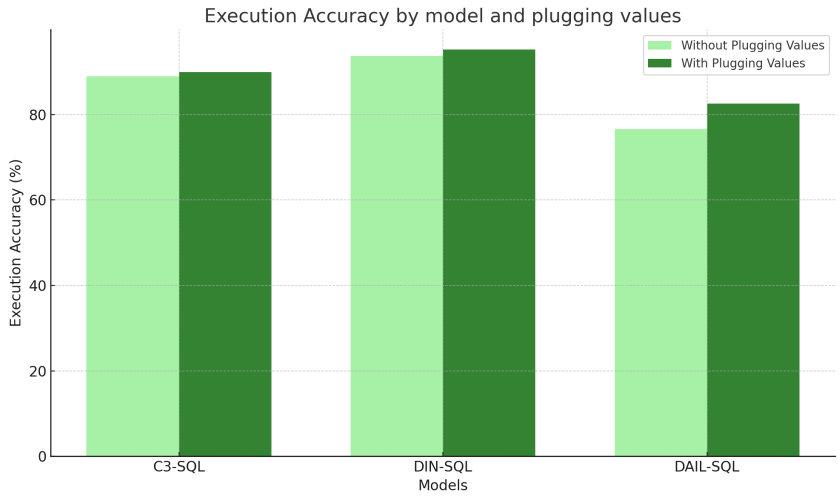
	easy	medium	hard	extra	all
count	42	80	37	42	201
========== EXECUTION ACCURACY ==========					
execution	1.000	0.912	0.649	0.643	0.826
========== EXACT MATCHING ACCURACY ==========					
exact match	0.952	0.675	0.405	0.214	0.587
----------PARTIAL MATCHING ACCURACY----------					
select	1.000	0.909	0.912	0.846	0.917
select(no AGG)	1.000	0.909	0.971	0.949	0.948
where	1.000	0.613	0.583	0.485	0.658
where(no OP)	1.000	0.613	0.583	0.667	0.711
group(no Having)	0.000	0.806	1.000	0.778	0.826
group	0.000	0.742	1.000	0.333	0.696
order	0.000	1.000	1.000	0.800	0.926
and/or	1.000	0.950	0.946	0.927	0.955
IUEN	0.000	0.000	0.667	0.333	0.429
keywords	1.000	0.859	0.706	0.692	0.812
---------- PARTIAL MATCHING RECALL ----------					
select	1.000	0.875	0.838	0.786	0.876
select(no AGG)	1.000	0.875	0.892	0.881	0.905
where	1.000	0.633	0.737	0.444	0.676
where(no OP)	1.000	0.633	0.737	0.611	0.730
group(no Having)	0.000	0.781	1.000	0.700	0.792
group	0.000	0.719	1.000	0.300	0.667
order	0.000	0.917	0.500	0.667	0.694
and/or	1.000	1.000	1.000	0.974	0.995
IUEN	0.000	0.000	0.286	0.125	0.200
keywords	1.000	0.824	0.649	0.643	0.771
---------- PARTIAL MATCHING F1 ----------					
select	1.000	0.892	0.873	0.815	0.896
select(no AGG)	1.000	0.892	0.930	0.914	0.926
where	1.000	0.623	0.651	0.464	0.667
where(no OP)	1.000	0.623	0.651	0.638	0.720
group(no Having)	1.000	0.794	1.000	0.737	0.809
group	1.000	0.730	1.000	0.316	0.681
order	1.000	0.957	0.667	0.727	0.794
and/or	1.000	0.974	0.972	0.950	0.974
IUEN	1.000	1.000	0.400	0.182	0.273
keywords	1.000	0.841	0.676	0.667	0.791

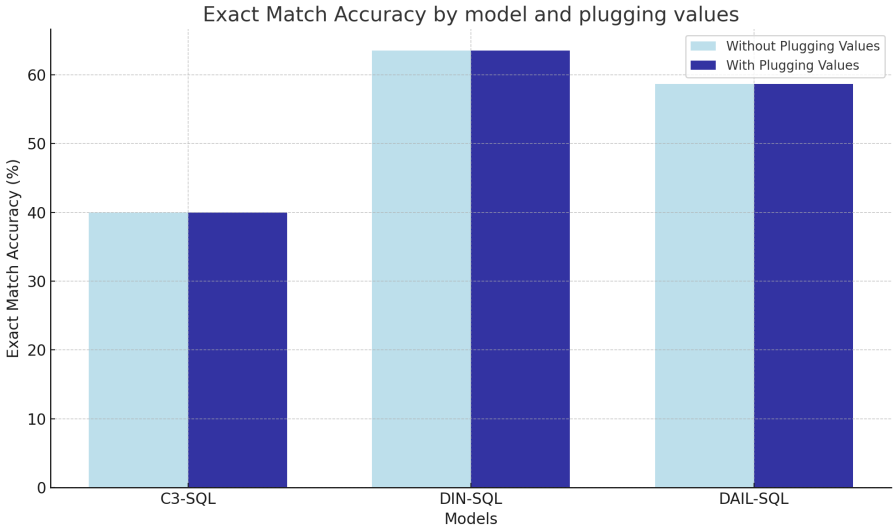
Fig 13.2: DAIL-SQL with plugging

7. Comparison

Following table compares the test accuracy and the accuracy breakdown for AGG, SEL and WHERE statements for baseline models - Seq2SQL and SQLNet.

Model	**Overall Test Accuracy**	**Accuracy Breakdown**		
		AGG	**SEL**	**WHERE**
Seq2SQL with RL	0.51	0.92	0.85	0.5
Seq2SQL w/o RL	0.51	0.92	0.85	0.5
SQLNet with col attention	0.58	0.9	0.9	0.6
SQLNet with col attention & embedding	0.65	0.91	0.91	0.68





LLM Models	Execution Accuracy		Exact Match Accuracy	
	Without plugging values	With plugging Values	Without plugging values	With plugging Values
C3-SQL with gpt-3.5-turbo	89.0%	90.0%	40.0%	40.0%
DIN-SQL with gpt-4	93.7%	95.2%	63.5%	63.5%
DAIL-SQL with gpt-3.5-turbo	76.6%	82.6%	58.7%	58.7%

The evaluation accuracy of the LLM models is shown in the table above. DAIL-SQL is the state of the art model at present. However, the original paper reports results using gpt-4 whereas we trained our DAIL-SQL using gpt-3.5-turbo which is why it provides accuracies that are lower. The model we trained using gpt-4 which is DIN-SQL achieved the highest execution accuracy scores as expected.

8. Challenges

We faced numerous challenges in this project. The challenges are briefly discussed below:

1. Obsolete dependencies with the old implementations - For the baseline models like Seq2SQL and SQLNET published in 2017, the dependencies were a huge problem to implement the baseline models. We faced the same issue even for the newer models like C3SQL, DINSQL, and DAILSQL. So all these errors had to be resolved to be able to proceed further with the implementation.
2. OpenAI API Tokens - The newer models like C3SQL, DINSQL, and DAILSQL used OpenAI APIs to establish a connection. This came with a risk of exceeding the tokens for the training. Because OpenAI API has a limit on using the number of tokens every day and there is also a limit on sending several requests every day this was a problem as we could not test it on the complete dataset and instead performed on fewer examples. The major error that we received was “Rate Limit Error” and for the number of tokens “Tokens per Day” and “Tokens per Minute”.
3. Cost associated with OpenAI APIs - There was also some cost associated with the OpenAI APIs.C3SQL and DAILSQL utilized the gpt3.5-turbo and DINSQL utilized the gpt4. Both of them cost differently. This was done so that we could test our model on a certain number of examples at the very least.
4. Computational Issues - There were various problems with the computation like memory and time. The time taken to run the samples was increased because of the OpenAI API requests. Because of this, it also took some time to understand what was happening and to wait for an accurate result. Another main issue was with memory. The dataset is huge which made it difficult to train the model.

9. Conclusion

In this project, we conduct a comparative study of the different models for the task of text-to-SQL conversion. We started with baseline models like Seq2SQL and SQLNET and implemented those. Further, we implemented the top 3 models for the Spider dataset namely C3SQL, DINSQL, and DAILSQL. Additionally, we implemented some hugging-face models for this task which would perform the task of converting the models further. Although there were multiple challenges along the way, we powered through it all and attempted to perform the task

of test-to-SQL conversion. Lastly, we made a User Interface where all the models come together to provide the SQL query generation of the corresponding natural language query.

10. References

[1] V. Zhong, C. Xiong, and R. Socher, "Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning," CoRR, vol. abs/1709.00103, 2017.

[2] X. Xu, C. Liu, and D. Song, "SQLNet: Generating Structured Queries From Natural Language Without Reinforcement Learning," arXiv preprint arXiv:1711.04436, 2017.

[3] X. Xu, "xiaojunxu/sqlnet: Neural network for generating structured queries from natural language.," GitHub, https://github.com/xiaojunxu/SQLNet (accessed Dec. 11, 2023).

[4] Dong, X., Zhang, C., Ge, Y., Mao, Y., Gao, Y., Lin, J. and Lou, D., 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv preprint arXiv:2307.07306.

[5] Pourreza, M. and Rafiei, D., 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. arXiv preprint arXiv:2304.11015.

[6] Gao, D., Wang, H., Li, Y., Sun, X., Qian, Y., Ding, B. and Zhou, J., 2023. Text-to-sql empowered by large language models: A benchmark evaluation. arXiv preprint arXiv:2308.15363.

[7] T. Yu 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.

[8] Brianm, "B-mc2/SQL-create-context · datasets at hugging face," b-mc2/sql-create-context · Datasets at Hugging Face, https://huggingface.co/datasets/b-mc2/sql-create-context (accessed Dec. 11, 2023).

[9] Bigbigwatermalon, "Bigbigwatermalon/C3SQL: The code for the paper C3: Zero-shot text-to-SQL with CHATGPT," GitHub, https://github.com/bigbigwatermalon/C3SQL (accessed Dec. 11, 2023).

[10] MohammadrezaPourreza, "Mohammadrezapourreza/few-shot-NL2SQL-with-prompting," GitHub, https://github.com/MohammadrezaPourreza/Few-shot-NL2SQL-with-prompting (accessed Dec. 11, 2023).

[11] BeachWang, "Beachwang/Dail-SQL: A efficient and effective few-shot NL2SQL method on GPT-4.," GitHub, https://github.com/BeachWang/DAIL-SQL (accessed Dec. 11, 2023).

[12] T. Yu, "Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task," *GitHub*, Dec. 11, 2023. https://github.com/taoyds/spider/tree/master (accessed Dec. 12, 2023).

11. Task Contributions

The tables below outline the contributions of each team member to the tasks.

Chapter	Report Contributor(s)
Introduction	Ankita Arvind Deshmukh
Objective	Ankita Arvind Deshmukh
Datasets	Ankita Arvind Deshmukh
Methodology	Ankita Arvind Deshmukh Sanika Vijaykumar Karwa Priyanka Birju Shah
User Interface	Priyanka Birju Shah
Results	Priyanka Birju Shah
Comparison	Sanika Vijaykumar Karwa Priyanka Birju Shah
Challenges	Sanika Vijaykumar Karwa
Conclusion	Sanika Vijaykumar Karwa
References	Ankita Arvind Deshmukh

Table 1.0: Project Report Contributions

	Code Sections	Code Contributor(s)
Models	Baseline Models	Ankita Arvind Deshmukh
	C3-SQL	Sanika Vijaykumar Karwa
	DIN-SQL	Priyanka Birju Shah
	DAIL-SQL	Ankita Arvind Deshmukh
	HuggingFace Models	Ankita Arvind Deshmukh Sanika Vijaykumar Karwa Priyanka Birju Shah
Evaluation		Priyanka Birju Shah Ankita Arvind Deshmukh
User Interface		Priyanka Birju Shah

Table 2.0: Code Contribution