Learning Schematic and Contextual Representations for Text-to-SQL Parsing

Prachi Jain ppjain@umass.edu **Vijayalakshmi Vasudevan** vijayalakshm@umass.edu **Nandhinee Periyakaruppan** nperiyakarup@umass.edu **Shebin Scaria** [*] sscaria@umass.edu

University of Massachusetts Amherst

1 Problem Statement

Motivation:

Deep contextual language models (Devlin et al., 2018), (Liu et al., 2019a), (Lewis et al., 2019),(Dong et al., 2019) (Raffel et al., 2019) have shown their effective modeling ability for text, achieving state-of-the-art benchmarks in a series of NLP tasks. These models capture the syntactic and semantic information of the input text, generating fine-grained contextual embeddings, which can be easily applied to downstream models. Despite the success of large-scale pre-trained language models on various tasks, it is less clear how to extend them to semantic parsing tasks such as text-to-SQL (Warren and Pereira, 1982), (Popescu et al., 2003), (Popescu et al., 2004), (Li et al., 2006), which requires joint reasoning of the natural language utterance and structured database schema information.

Recent works (Guo et al., 2019), (Wang et al., 2020b), (Bogin et al., 2019a), (Bogin et al., 2019b) shows that highly domain-specific semantic parsers can be further improved with more powerful pre-trained language models, even though these language models are trained for pure text encoding. However, based on error analysis on the output of neural language model-based text-to-SQL systems, it has been observed that these models can be further enhanced if the following three pain points could be mitigated:

1. **The model is ineffective in matching and detecting column names in utterances.** It should learn to detect such column names by matching utterance tokens with the schema and using the matched columns in the generated SQL. The error analysis indicates that, models miss some columns when synthesizing the target SQL, while the column is mentioned explicitly in the utterance.

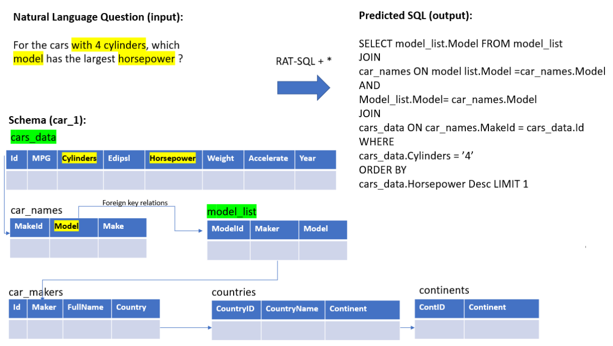
2. **The model fails to infer the columns implicitly from cell values.** This problem is trickier than the first one because the model is expected to infer the column name based on some cell values mentioned in the utterance instead of just matching the utterance tokens with the schema. This requires the model to have more domain knowledge.

3. **The model should learn to compose complex queries.** Besides the column selection, the model should learn to attach the selected columns to the correct clauses to generate a correct SQL. This is a non-trivial task, especially when the target SQL is complex, e.g., when the query is nested.

Goal Recent works have demonstrated that jointly pre-training on utterances and table contents (e.g., column names and cell values) can benefit downstream tasks such as table parsing and semantic parsing (Yin et al., 2020) (Herzig et al., 2020). It is hypothesized that to cope with the three pain points previously listed, pre-training objectives must be used to enforce the learning of contextual representations that better capture the alignment between utterances and schema/table contents.

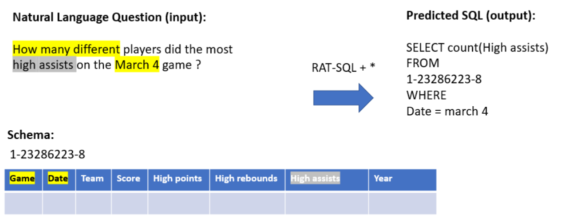
With the intent to explore the area of query generation from natural language, we chose Relation-Aware Schema Encoding and Linking for SQL (RAT-SQL) (Wang et al., 2020a) framework and other similar work such as GAP with RATSQL (**?**), GRaPPA with RATSQL (Tao Yu, 2021). Thus, we intend to setup a framework such as given a query and the schema, the model should predict a query like mentioned in Figure 2.

[0]Written in an order so that all names fit; Code: https://github.com/nirvana1707/text-to-sql



```
SELECT model_list.Model FROM model_list
JOIN
car_names ON model list.Model =car_names.Model
AND
Model_list.Model= car_names.Model
JOIN
cars_data ON car_names.MakeId = cars_data.Id
WHERE
cars_data.Cylinders = '4'
ORDER BY
cars_data.Horsepower Desc LIMIT 1
```

Figure 1: Text to SQL task being inferred using RAT-SQL model with a complex underlying schema taken from Spider dataset. Here, * refers to different embedding architectures. The inference/predicted SQL mentioned in the figure comes from RoBERTa prediction. The highlighting indicates the relationship between the input tokens of the natural language and the underlying schema. We see this detail in the section of Alignment Loss



```
SELECT count(High assists)
FROM
1-23286223-8
WHERE
Date = march 4
```

Figure 2: Text to SQL task being inferred using RAT-SQL model with a simpler underlying schema consisting of only 1 table taken from WikiSQL. Here, * refers to different embedding architectures. The inference/predicted SQL mentioned in the figure comes from RATSQL+GloVe prediction. The highlighting indicates the relationship between the input tokens of the natural language and the underlying schema. Here grey coloring indicated the a relationship between this token was expected to be captured by the alignment matrix in 11, but was too low

Problem Formulation: We formulate the problem statement similar to the RAT-SQL framework(Wang et al., 2020a). A natural language question $\mathcal{Q}$ is given as input along with a schema which encompasses of columns and table. The task will be to generate SQL query which will answer the question for the user. The question $\mathcal{Q}$ can be visualized as sequence of words and schema will consist of columns $\mathcal{C}$ and tables $\mathcal{T}$. Each column name and table name can be visualized as sequence of words. The SQL query ($\mathcal{P}$) is represented as an abstract syntax tree. Few columns in the schema are called primary keys which uniquely identifies a record in the table and there will be foreign keys too. Each column type can be classified as a number or text. For modelling the text-to-SQL problem, an encoder-decoder framework is incorporated where the question contextualized schema graph $\mathcal{G}_Q$ is passed as input which is then converted by the encoder to an embedding with joint representation c_i, t_i, q_i where $c_i \in \mathcal{C}, t_i \in \mathcal{T}$ and $q_i \in \mathcal{Q}$. The decoder then uses this embedding to compute the conditional probability over all SQL programs given the input embedding.

2 What you proposed vs. what you accomplished

As per our initial proposal, we initially set out to the do the below, and made changes along the way as mentioned below:

- ~~Setup RATSQL codebase from scratch~~
  - We failed to develop the codebase due ro its complexity and GPU/RAM resource constraints. We found mismatch in the resource specifications mentioned in the codebase.
  - We tried numerous approaches for approximately a month and reached out to the instructors for guidance
  - Instead we focused on a comparative analysis of similar text-to-query models by taking implementations from GAP, and GRaPPa.
- Replicating results of baseline models mentioned in RATSQL:
  - We trained the RAT-SQL+GloVE on WikiSQL dataset and RAT-SQL+GloVe, BERT baseline models mentioned in RAT-SQL
  - As mentioned above, we followed a similar pipeline for GAP and GRaPPA
- Evaluation Metrics: As mentioned in our proposal, we setup Logical Form and Execution Accuracy. Additionally, we worked on component matching and query complexity categorization
- Experiments:
  - ~~Implementing a ranker module for top candidate selection~~ Due to time limitations we could not complete it.

    – Instead we focused on human human evaluation/prompting/rewritten utterance based experiments

- Error Analysis: Though not part of our initial proposal, we analyze and explored different types of errors generated by the model

All our work is added to the publicly available git repository[1].

3 Related work

We look at **RATSQL + Grammar-Augmented Pre-Training** (Tao Yu, 2021), that works well for table semantic parsing exclusively for relational database. This model outperforms RAT-SQL+BERT based on the paper, as BERT is trained on Wikipedia data and large book corpus whereas GraPPa is trained on tabular data and synthetic text-to-SQL data from popular semantic parsing datasets like Spider. This methodology distinguishes GraPPa from the RAT-SQL+BERT/GloVE models that we selected. The language model is pre-trained using masked language modelling loss and SQL semantic loss.

The model **RATSQL + GAP** (Peng Shi, 2021) ”Generation Augmented Pre-training Approach” attempts to mitigate the issue of inferring the column names in the schema. It does this by labelling the schema, checking in the input and replacing column names by values and validating where the model is interpret it. Our approaches tries to encoding representations in the graph constructed in the schema, but is not as tightly as the above. Hence, we add this as part of our comparative analysis of text-to-query prediction to analyze whether the approach actually generates better results, or is able to generalize over test set or new inferences.

We see that there are several pre-trained language model that be used as a contextualized embedding which can be fed as input to RATSQL transformer. One such PLM is RoBERTa(Liu et al., 2019b). We see that use of **RoBERTa** has several advantages over BERT as stated in this study. This study(Liu et al., 2019b) addresses the challenges of comparing different approaches in language model pretraining. It acknowledges that training language models is computationally expensive, uses private datasets of varying sizes, and emphasizes the significant impact of hyperparameter choices on the final results. The study focuses on replicating the BERT pretraining approach and carefully examines the effects of various key hyperparameters and training data size. The findings indicate that BERT was undertrained and can achieve performance that matches or surpasses subsequent models.

The paper **GPTScore: Evaluate as You Desire** (Fu et al., 2023) proposes a novel evaluation framework that uses generative pre-trained models to score generated texts based on natural language instructions. It explores 19 pre-trained models of different sizes and evaluates them on four text generation tasks, 22 evaluation aspects, and 37 datasets. Some of its exciting ideas are introducing a flexible and customizable way of evaluating text generation without annotated samples or predefined metrics. Next, leveraging the zero-shot instruction and in-context learning abilities of large pre-trained models to capture users' true desires and expectations for text quality. Finally, it conducts extensive experiments and analyses on various text generation tasks and evaluation aspects and comprehensively compares different pre-trained models. However, It relies on conditional generation probability as the sole criterion for scoring texts, which may not capture all the nuances and subtleties of natural language. The paper was helpful in setting up a human evaluation task for Spider Dataset. It provided insight and reasoning for how the evaluation protocol and aspect definition can be designed or selected for each task and aspect.

We refer to the work done (Yu et al., 2018a) to setup our **SQL hardness criteria** by classifying SQL queries based on different levels of complexity. We follow the same structure to ascertain and standardize the levels of difficulty along with component matching to determine the syntactic similarity of generated queries with gold query.

In the work done by (Zhong et al., 2017), we understand the Seq2SQL models for generating SQL queries from natural language questions. It use a sequence-to-sequence architecture with attention and copy mechanisms to generate SQL queries and reinforcement learning to optimize the query execution accuracy on a database, using rewards from query execution as feedback signals.We understand the concept of **execution**

[1]https://github.com/nirvana1707/text-to-sql

accuracy and logical form accuracy and how we can extend it to the WikiSQL Dataset for our project and also observe the pros and cons of each approach. We see scenarios where a query's result could be correct, but its differences could help understand these queries as different.

Our initial proposal mentioned **RNG-KBQA: Generation Augmented Iterative Ranking for Knowledge Base Question Answering** (Ye et al., 2021), a Rank and Generate based approach for question answering on knowledge base. We wanted to use the idea to help improve the model with better choices of SQL query generation handle generalization scenarios since it is difficult to generate for an unseen database schema.

As one of our key components is the study on evaluation metrics, we went through **'Improving text-to-sql evaluation methodology'**(Catherine Finegan-Dollak, 2018), which proposes seven different types of text-to-SQL data sets to test the model on, which will help them evaluate the model's strength and weaknesses. They also discuss about the issue that current division of data into training and test data set doesn't completely test how these systems generalize to new queries. Hence, they propose a contemporary data set split for future work in this area. In this split no SQL query is allowed to occur in more than one set.

We also refer to **'A survey on deep learning approaches for text-to-sql'** (Katsogiannis-Meimarakis, 2023) that looks at several pretrained model based approaches for text to query generation models. The work explores different evaluation methodologies, pros/cons of each approach and challenges involved in order to improve the accuracy are highlighted. We use this as a reference to look at our results and possible way forward.

In Evaluating the Text-to-SQL Capabilities of Large Language Models(Rajkumar et al., 2022), we see that they evaluate the Text-to-SQL abilities of the Codex (deprecated now, we are using text-davinci-003 engine for this experiment) language model through empirical analysis. The results show that Codex performs well as a baseline on the Spider benchmark, even without any finetuning (inference only approach). Additionally, by providing a small number of in-domain examples in the prompt, Codex outperforms state-of-the-art models that have been finetuned on similar few-shot examples on the GeoQuery and Scholar benchmarks.
In Self-Ask (Press et al., 2022), we see a new prompting based method for LLMs where it tries to improve the model's ability to decompose large queries. We tweak this idea to run an experiment and observe the results when we give large and interaction based questions/queries.

4 Dataset

Our experiments are based on the below datasets:

4.1 Spider

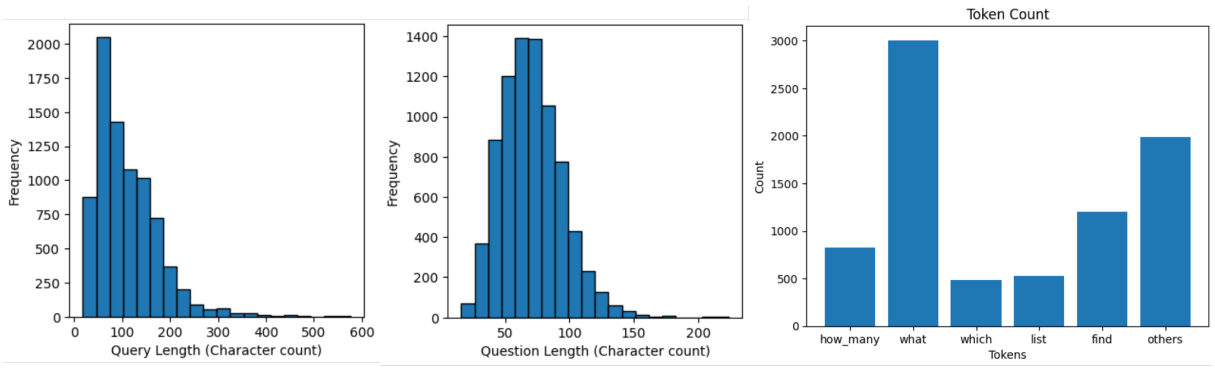
4.1.1 Overview

(Yu et al., 2018b) dataset is a large-scale cross-domain semantic-parsing and text-to-SQL benchmark. Spider consists of 7,000 text-to-SQL pairs for training, 1,034 pairs for validation, and 2,147 pairs for testing. These text-to-SQL pairs span across 200 complex databases in 138 domains. Spider resembles real-time databases that contain multiple tables and include complex relationships between the different tables.

We also perform Exploratory Data Analysis (EDA) on the spider dataset. For our analysis, we consider the 7000 training and the 1034 validation text-to-SQL pairs. Figure 3 (a) shows the distribution of the query sizes in the dataset. From this plot we can observe that the on average, a SQL query is of 100 characters length. Figure 3 (b) shows the distribution of the question sizes and Figure 3 (c) shows the distribution of the questions in the dataset. From our analysis, we can also observe that majority of the question types are 'How many', 'What', 'Which', 'List', and 'Find', indicating that Spider contains realistic data.

4.1.2 Data preprocessing

Our experiments follow two different preprocessing strategies for the spider dataset. For some of our experiments, we use Stanford Core NLP toolkit for performing pre-processing. To use stanford core NLP, we establish a connection to the corenlp server from our python environment, which allows us to send requests and retrieve processed results. For experiments where we faced difficulties in setting up a connection to



(a) Distribution of query sizes

(b) Distribution of question sizes

(c) Distribution of questions

Figure 3: Exploratory Data Analysis on spider dataset

the corenlp server, we switched it with Stanza, a Python library built on top of the CoreNLP library. By using Stanza, we could simplify the setup process and directly leverage the power of CoreNLP in our Python environment.

4.2 WikiSQL

4.2.1 Overview

We used the **WikiSQL** (Zhong et al., 2017) dataset, which is a relatively simpler dataset as it does not pose the challenge of including relationship between multiple tables. The WikiSQL dataset consists of 56,355 text-to-SQL pairs for training, 8,421 pairs for validation and 15,878 pairs for testing.

4.2.2 Data preprocessing

For processing the WikiSQL dataset, we use Stanford Core NLP as mentioned in section 4.1.2. Since the WikiSQL dataset is much smaller in size, as compared to the spider dataset, we were able to run the pre-processing using Stanford Core NLP library smoothly.

4.3 SParC

4.3.1 Overview

In addition to the above, we also used **SParC** (Yu et al., 2019).

We selected the SParC dataset primarily for testing purposes, with the intent to check the generalisability of the model on an unseen dataset. Here, the constraint was the dataset should have queries that use similar schema. Our explorations lead to the SParC dataset which uses the same databases as the Spider dataset. The utterance, query pairs from the dataset are of an interaction nature, an example has been provided in our analysis section 8. The dataset consists of ”interaction” questions to retrieve the answer of a common objective question referred with the key ”final” in the dataset (along with the gold sql query). There are 166 databases as part of the dataset. Here we only take the DEV dataset provided (dev.json file), which consists of 422 of ”final” questions 1203 ”interaction” questions, with 2 to 4 ”interactions” per ”final” question. We represent a sample input output pairs in Table 8.

4.3.2 Data preprocessing

We preprocess the SParC dataset to extract only the ”interaction” based utterances, its corresponding gold queries and other attributes. In addition we convert it into Spider's equivalent format by restructuring the json to have all the right attributes. We do this to reuse the existing tokenization pipeline for this dataset. We do not consider the ”final” question as part of this because the key lacks certain attributes such as ”sql” which is a logical form of representing the query, which is only available for ”interaction” questions. However we use the ”final” question to run our experiment on question/sub-question answering as part of the inference pipeline.

5 Methodology:

As part of preprocessing the natural language corpora of queries $\mathcal{Q}$,schema columns $\mathcal{C}$,schema tables $\mathcal{T}$, we plan to perform tokenization and lemmatization using standard NLP toolkits.

We will use the pre-trained BERT s for obtaining the initial representation to the encoder model. The main objective of the designed encoder should be able to 'encode' relationships between the queries and schema. The encoder then applies a stack of N relation aware self-attention layers. The final layer representation of the Nth layer will constitute the whole output of the encoder. Schema linking is done to help the model align the column or table references in the question to the corresponding schema columns or tables. Value based linking and a name-based linking is done to obtain this correspondence. Value based linking relationship is done to extract values of a column to the column name e.g. in Figure 2, column 'model' will be associated with this values like "BMW", "Honda" etc. Whereas, name based linking will have a relationship associating 'exact' or 'near exact' column/table name alternatives e.g. linking 'car manufacturer' with 'car_makers' table.

6 Baselines

We intend to use the RAT-SQL and RAT-SQL+BERT (Wang et al., 2020a) as our baseline algorithms, as we attempt to replicate, reproduce the results and compare the same with the provided results.RAT-SQL is a relation-aware semantic parsing model that generates SQL queries from natural language questions. It works as follows:

- It uses a relation-aware self-attention encoder to encode the natural language question and the database schema. The encoder captures the relations between database columns and their mentions in the question and produces a contextualized representation for each token.
- It uses a sequence-to-sequence decoder with attention and copy mechanisms to generate the SQL query from the encoded representation. The decoder also leverages the structure of SQL to prune the output space and simplify the generation problem.
- It uses reinforcement learning to optimize the query execution accuracy on the database, using rewards from query execution as feedback signals. It also uses cross-entropy loss to pre-train the model on the gold queries.

Futher the flow can be explained in 5. During preprocessing, the input of questions, column names, and table names are tokenized and lemmatized with the StandfordNLP toolkit . Within the encoder, **RAT-SQL uses GloVe** word embeddings, held fixed in training except for the 50 most common words in the training set. For **RATSQL BERT**, WordPiece tokenization is used. All word embeddings have a dimension of 300. The bidirectional LSTMs have a hidden size of 128 per direction and use the recurrent dropout method with a rate of 0.2. The baseline stacks eight relation-aware self-attention layers on top of the bidirectional LSTMs. The following parameters are set within them as dx = dz = 256, H = 8, and use dropout with a rate of 0.1. The position-wise feed-forward network has an inner layer dimension of 1024. Inside the decoder, rule embeddings of size 128 are used, node type embeddings of size 64, and a hidden size 512 inside the LSTM with a dropout of 0.21.

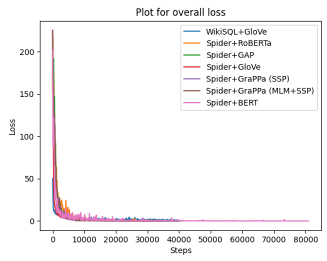
The baseline algorithms were performed as:

- **RAT-SQL + GloVe** on WikiSQL (train/dev split ratio as 56355/8421) and Spider (train/dev split ratio as 8659/1034) with max-steps = 40k, batch size = 10 and Adam optimiser with learning rate = 1e-3
- **RAT-SQL augmented on pre-trained BERT** for Spider(train/dev split ratio as 8659/1034). max-steps = 81k, batch size = 4, and adam optimiser with learning rate = 1e-4 and bert learning rate: 1e-5. Note: we faced performance issues while training the model with BERT and hence used the RoBERTa's hyperparameters to train and get competitive the performance.

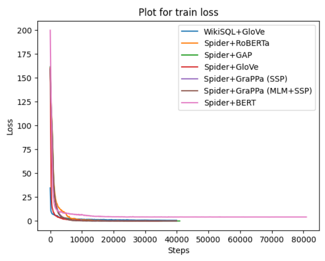
Reasoning for chosing these baselines over over other models: RAT-SQL (v3+ BERT) is the SoTA model with one of the highest accuracy on the Spider leadeboard that also provides an open-source implementation. It adds string matching to the encoder through the use of relation-aware self-attention and adopts a tree-based decoder to ensure the correctness of the generated SQL.

7 Model Architectures

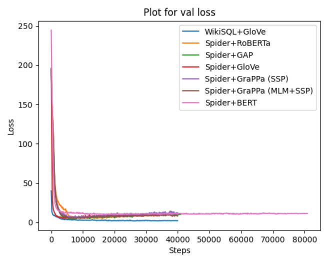
- **RATSQL + RoBERTa** RoBERTa is a pre-trained language model that improves upon the existing BERT model by using a combination of GPT's tokenizer algorithm and BERT's encoder blocks. It is trained on a larger dataset of 160 GB (Book Corpus + Wiki) using the Byte pair encoding algorithm



(a) Overall Loss Plot



(b) Loss Plot for Train



(c) Loss Plot for Val

Figure 4: Loss Plots

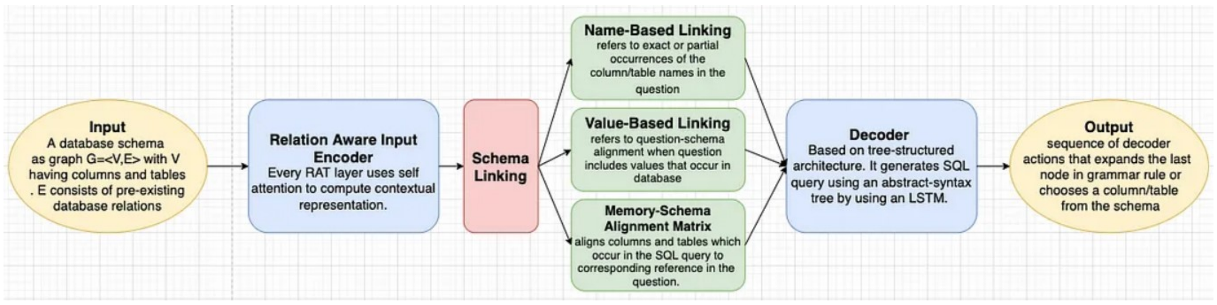

Input
A database schema as graph G=<V,E> with V having columns and tables . E consists of pre-existing database relations
Relation Aware Input Encoder
Every RAT layer uses self attention to compute contextual representation.
Schema Linking
Name-Based Linking
refers to exact or partial occurrences of the column/table names in the question
Value-Based Linking
refers to question-schema alignment when question includes values that occur in database
Memory-Schema Alignment Matrix
aligns columns and tables which occur in the SQL query to corresponding reference in the question.
Decoder
Based on tree-structured architecture. It generates SQL query using an abstract-syntax tree by using an LSTM.
Output
sequence of decoder actions that expands the last node in grammar rule or chooses a column/table from the schema

Figure 5: A flow chart of RAT-SQL model

for tokenization, and the optimization is performed at the data level. RoBERTa uses dynamic masking during training, copies the dataset 10 times, and uses a different mask in each copy. The hyper-parameters used in RoBERTa include Adam optimizer, learning rate, dropout, training steps, batch size, and input length. RoBERTa provides a more refined and robust version of the BERT model with improvements in the model architecture, training strategy, and data processing. Integrating this with the RAT+SQL architecture makes the contextualized embedding that gets passed into RATSQL richer as compared to BERT.

- **RATSQL + GAP** Generation Augmented Pre-training (GAP) is a model pre-training framework that jointly learns representations of natural language utterances and table schemas by leveraging generation models to generate pre-train data. GAP Model is trained on 2M utterance-schema pairs and 30K utterance-schema-SQL triples, whose utterances are produced by generative models. It uses two generative models to create pre-training data: a schema-to-utterance model that generates natural language questions given a table schema, and an utterance-schema-to-SQL model that generates SQL queries given a question and a schema. It uses three pre-training tasks to train a text-to-SQL encoder: masked language modeling (MLM), schema linking, and SQL generation. MLM predicts the masked tokens in the question or the schema; schema linking predicts the alignment between the question tokens and the schema tokens; SQL generation predicts the SQL query given the question and the schema. Based on experimental results, neural semantic parsers that leverage GAP MODEL as a representation encoder obtain new state-of-the-art results on Spider benchmarks.

- **RATSQL + GraPPa (MLM + SSP)** GRAPPA, is a pre-training approach for table semantic parsing that aims to learn a compositional inductive bias in the joint representations of textual and tabular data. It constructs synthetic question-SQL pairs using a synchronous context-free grammar (SCFG) over high-quality tables. GRAPPA is initialized with RoBERTaLARGE and further pre-trained using synthetic data with an SQL semantic loss and table-related data with an MLM loss. In the MLM objective, the natural language sentence and table headers undergo masking. A portion of the input sequence is replaced with the ¡mask¿ token, and the MLM loss is computed by predicting the masked tokens. Default hyperparameters from (Devlin et al., 2018) are followed, including a 15% masking probability. The MLM + SSP uses SSP objective (which provides an auxiliary task that improves column representation training by predicting column presence and associated operations in SQL queries) in addition with the above described MLM approach. The authors of GRaPPa believe that this combination enhances the model's understanding of both language and semantic structures, leading to better performance in tasks involving natural language understanding and SQL query generation.

- **RATSQL + GraPPa (SSP)** The SSP (SQL Semantic Prediction) objective introduced in GRaPPa, is an auxiliary task added to train column representations in the context of their work on natural language sentence and SQL query. This aims to predict whether a column appears in the SQL query and determines the operation triggered for that column. Given a natural language sentence and table headers, the objective involves classifying each column based on its presence in the SQL query and the corresponding operation.pairs. By adding the SSP objective, the authors aim to train column representations and enhance the model's understanding of the relationship between natural language sentences, table headers, and SQL queries.

8 Experiments

8.1 Pretrained Models

Using RAT-SQL as baseline algorithm with other model architectures such as RoBERTa (Liu et al., 2019c), GraPPa (Tao Yu, 2021) , GAP (Peng Shi, 2021)

Model architecture and inputs, Train-Validation split, Training Plots, Hyperparameters, Other Training Settings - GPU/Colab/-Docker etc

The cross-domain datasets Spider, WikiSQL evaluate models in a zero-shot setting, meaning the model is trained on one set of domains and evaluated on a completely disjoint set.

- **RATSQL for WikiSQL** The implementation for this task was using the instructions given in the RATSQL repository by microsoft. However, we ran into few issues with respect to SQLLite while executing the dbengine.py file as part of pre-processing step. We had to downgrade the version of SQLAlchemy and protobuff for successful implementation of training. We worked on this training the model for 40000 steps on our local computer which had 16GB RAM. The training took about 3 days for us. We used a docker container based implementation for performing this experiment.

- **RATSQL for Spider** The implementation for this task was using the instructions given in the RATSQL repository by microsoft. However, we ran into few issues with respect to SQLLite while executing the dbengine.py file as part of pre-processing step. We had to downgrade the version of SQLAlchemy and protobuff for successful implementation of training. We worked on this training the model for 40000 steps on our local computer which had 16GB RAM. The training took about 4.5 days for us. We implemented this directly on a local system for performing this experiment.

- **RATSQL + BERT for Spider** The approach that we took was to use the embedding generated by pretrained language model BERT (path in our git repo: main/ratsql/models/spider/spider_enc.py). This method failed due to memory constraints, exhaustive environment setup requirements and limited computational resources and memory. We modified the implementation for pre-processing where we replaced coreNLP with stanza as we faced time out errors while setting coreNLP. The entire experiment was run on 1 dedicated GPU(gypsum-m40) with 250 GB memory based environment. Finally, since BERT was not performing as reported by RATSQL authors we used the hyper parameters of RAT-SQL plus Roberta to expect competitive performance. Finally, we feel by using such hyper parameters the model becomes biased and is not generalisable to other datasets.

- **RATSQL + RoBERTa for Spider** The approach that we took was to replace the

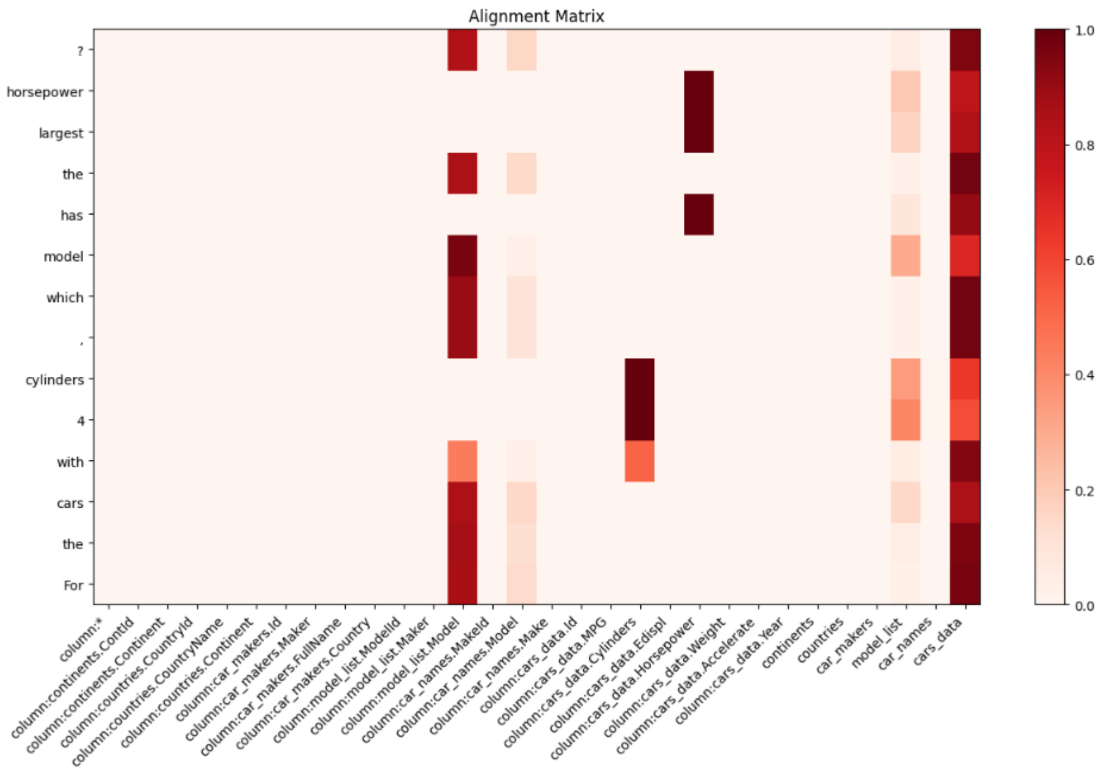


Figure 6: Alignment matrix for a question, inferred on RoBERTa. The sample utterance has been taken from Spider dataset. y-axis corresponds to the tokens extract from the utterance (in the order that start of the token is at the origin) while the x axis represents the schema. From the right of the x-axis, we see the tables: cars_data, car_names, model_list, car_makers, countries, continents belonging to the database car_1, while the remaining represent columns. Columns have been named in the format - "column":table_name.col_name where col_name belongs to the table table_name

embedding generated by pretrained language model BERT to ROBERTa(path in our git repo: main/ratsql/models/spider/spider_enc.py). We expect this method also to fail in similar ways as that of BERT but, since RoBERTa is a more robust version as compared to BERT due to its architectural nuances and the way it is being trained, we expected the results to improve. This work was witnissed in most of the paper related to semantic parsing and we did refer few repositories to get the hyper-parameters that we trained RAT-SQL+RoBERTa (we used the RoBERTa-large pretrained model from transformers library) on. Since this is a huge model (training bert itself required a GPU with atleast 16GB memory), we could not perform hyper parameter tuning using grid search as it was not computationally feasible. Hence, we went ahead using hyper-parameters(path in our git repo: main/experiments/spider-roberta-run.jsonnet) given in the existing codebase. We also used modified implementation for pre-processing where we replaced coreNLP with stanza as we faced time out errors while setting coreNLP on colab. The entire experiment was run on google Colab. We faced some issues while running pre-processing step and had use the file given in the repo as that reported lower loss when compared to the file that we generated on our own. There were issues with storage of model checkpoints and google drive quota exceeded error. Hence, we had to buy additional storage units to store each checkpoint which was 5GB long and the entire training took about a week as there were plenty of interruptions either due to time-out or quota exceed error. Also, we faced issue in loading the pre-trained checkpoint wherein we got an unexpected key error. Hence, we had to work with saver.py (path in our git repo: main/ratsql/utils) file in the repository to handle this error which was caused due to the version of pytorch used in colab. We purchased additional compute units as well for training the model. We had to stop 34100 iterations as we got a better accuracy as comapred to baseline and we were running out of storage. Model performed better with few improvements in the errors that occured in the baseline.

- **RATSQL + GAP for Spider** The approach that we took was to use the embedding generated by pretrained language model GAP (path in our git repo: seq2struct/models/spider/spider_enc.py).The entire experiment was run on 1 dedicated GPU (gypsum-m40) with 250 GB memory based environment. Finally, GAP performs at par accuracy with what its authors reported in their paper.
- **RATSQL + GraPPa (MLM + SSP) for Spider** Implementing RATSQL + GraPPa (MLM + SSP) involved two steps. First was to pre-train Grappa on MLM + SSP objective. Second was to train RATSQL + GraPPa (MLM + SSP) on the spider dataset. Since the pretraining process took 8 16GB Nvidia V100 GPUs for the authors of Grappa, and we could not afford such heavy resources, we utilized the pretrained grappa checkpoints as provided in their repository. RATSQL + GRaPPa codebase is built on top of the RAT-SQL repository. We included changes in this to incorporate the pretrained Grappa MLM + SSP checkpoint before starting the training. The entire experiment was run on 1 GPU based environment. We ran the training for 40000 steps. Since the unity cluster restarted in the middle of the training after an outage, we ran the training in two parts. The total training time was around 120 hours. RATSQL + GraPPa (MLM + SSP) performed significantly better than the baselines when tested on logical form accuracy. This variant of RATSQL outperformed the other variants when tested on the execution accuracy.
- **RATSQL + GraPPa (SSP) for Spider** Similae to implementing RATSQL + GraPPa (MLM + SSP), implementing the RATSQL + GraPPa (SSP) also involved two steps. First was to pre-train Grappa on SSP objective. Due to resource constraints, we utilized the pretrained grappa checkpoints for SSP objective as provided in their repository. We then included changes to the RAT-SQL codebase to incorporate the pretrained Grappa SSP checkpoint before starting the training. he entire experiment was run on 1 GPU based environment. We ran the training for 40000 steps and the total training time was around 110 hours. RATSQL + GraPPa (SSP) performed significantly better than the baselines when tested on both logical form accuracy and execution accuracies.

8.2 Few shot and Zero Shot prompting for Spider

Given the complexity of the training that we faced by performing above experiments, we wanted to explore techniques which involved minimal training. We tried perform few-shot prompting to convert text to SQL query and we constructed the prompts as suggested in in this work(Rajkumar et al., 2022). The zero shot setting gave us an accuracy 51% wherein the prompts that we constructed for each question comprised of just the schema details along with the question to be answered. We then moved to a few shot setting wherein we gave about 16 examples randomly chosen from easy, medium, hard and extra hard queries from spider training dataset. But we made sure that there are more number of medium and hard queries as compared to easy ones because easy ones gave us higher execution accuracy in case of zero-shot itself. As we gave few example question+query pair and schema details for the question that was asked, few shot setting gave us about 64.3% accuracy as seen in Table 4. We used text-davinci-003 engine called using openai create api. We used openai free credits to do this experiment and if we had to stop the experiment at this stage as we did not have more credits. Since we have by far looked at trained model, while doing a inference only task, we computed the execution accuracy instead of exact match or logical form accuracy as the inference only prompting based task works with a LLM which learns the instruction rather than patterns in the query structure. It is already equipped with SQL to produce query. The prompting examples are to make the LLM understand the different questions asked and how it is mapped to a query.

8.3 Human-aligned evaluation with GPT-3.5 and GPT-4

Task: Human evaluation remains the gold standard in ML research. Can we leverage GPT-3.5

capabilities to create an evaluation metric for Text-to-SQL that aligns with human preference?
Relevance: The problem with our existing evaluation metrics for Text-to-SQL models is they need to accurately measure human preference. For example, the evaluation metrics in various Text-to-SQL SoTA reject fine SQL queries for not having the same strict execution result as golden (aka. labeled) queries, although most of the queries generated were actually perfectly fine from a human perspective.
In another instance, the standard evaluation metric for document summarization benchmarks is ROUGE, which is based on counting overlapping word phrases between a generated summary and a golden summary. This approach fails to detect correct summaries written in different terms and phrases from the golden summary. Despite this, ROUGE scores are still being used by recent papers.

Methodology:
A Text-to-SQL task entails:

- An input question in natural language text.
- Generating a SQL query.
- Executing the SQL query and obtaining structured data (i.e., rows).

To evaluate a generated SQL query, we can prompt GPT-3.5 to answer the input question using the structured data (Spider) result from executing the query. This will give us a natural language answer to the input question. We repeat this step for the golden SQL query. Then, we ask GPT-3.5 whether the hypothesis answer generated from the generated query is correct, given the input question and the reference answer generated from the golden query's result. Finally, we receive a binary Yes or No response from GPT-3.5. By using GPT-3.5 in this way, we can create an evaluation metric for Text-to-SQL that better aligns with human preference.

8.4 Oracle Rewritten Utterance

Task: To check if utterances which suffered from exact match accuracy could be rewritten to generate queries that are more aligned with the column and tables in the relevant schema.
Relevance: A key model property to investigate would be its ability to link tokens in the natural language sentence with the underlying schema. It would be interesting to analyze this parsing capability of the model which includes the preprocessing-annotation pipeline, ability to the encode the meaning and representation of the tokens or its ability to map these with the schema structure to generate the final query. Thus, one possible way to check this is to rewrite or paraphrase the utterance, pass it to model and analyze whether this processed utterance is able to generate results with better syntactic match with the gold SQL query.

Methodology: We extend the idea of oracle-rewritten utterance (Li et al., 2023) to setup this experiment. Here as input we select few samples from the Spider's Dev dataset (i.e. dev.json) filtered based on the the results obtained from RoBERTa predictions to select only those examples which suffers from exact match accuracy i.e. where the predicted query is not an exact match with the gold sql query. To implement this idea, we extend our inference pipeline using our RoBERTa checkpoint 34100, and generate predicted queries given an utterance. For rewritten utterances, we manually attempt to rewriting and paraphrase these utterances.

8.5 Question/Sub-question Experiments using SParC

Task: Here, we intend to address the question whether the selected text to query generation models are able to answer complex questions ?
Relevance: We extend the core idea of the self-ask paradigm (Press et al., 2022) to our project. Though the proposed methodology is primarily for large language models, we use this idea to evaluate and answer questions such as does the model handle complex questions, , does it help if this question is decomposed to subquestions, or are there any underlying issues plaguing the model to tackle these ?
Methodology:
We take samples from SParC (Yu et al., 2019) dataset which follow the paradigm of question-subquestions. We earlier mentioned about the preprocessing steps for SParC to test the model on new datasets which involved only subquestions/interaction based questions. Here we do not restrict ourselves to these, instead we consider the main question that is asked and fire follow-up questions

to look at execution accuracy to determine whether the model is able to generate these results. Thus, we also intend analyze the results obtained from SParC in terms of interaction.

9 Evaluation Metrics

We have evaluated the selected models on numerous evaluation metrics that are pertinent for text to sql query conversion. We list down evaluation metrics used as below:

String matching (or Logical Form Accuracy) (Zhong et al., 2017) is the simplest accuracy metric for text-to-SQL parsing. This metric evaluates the similarity between the ground truth SQL query and the predicted SQL query by treating them as strings and checking if they are identical. In string matching, an exact match is only counted when the predicted query exactly matched the ground truth query. This metric, however, has some limitations, in that it penalizes valid queries that are semantically equivalent. To overcome this, we use execution accuracy that assesses the execution results of the query instead.

Execution accuracy (or Query Accuracy) (Zhong et al., 2017; Yu et al., 2018a) is another simple approach for comparing SQL queries. It involves executing both the ground truth and predicted SQL queries against the corresponding database or table and comparing the results they return. If the results are the same, the prediction is considered correct. One downside of this metric is that, it is subjected to false positives when the predicted query does not correspond to the question but produces desired results. For example the following two queries 'SELECT Name, Age FROM Employees WHERE Department = HR' and 'SELECT Name, Age FROM Employees WHERE Age > 20' may produce the same results if all the HR employees are aged above 20. Due to this, we consider the execution accuracy metric with the string matching metric.

Component Matching (Yu et al., 2018a) is an approach that is used to assess the different parts or components within the SQL query. It provides a better understanding of which parts of the predicted query are correct. For example, if we want to compute the select column accuracy, we compute the percentage of the predicted queries that have the same columns in the SELECT as the ground truth queries. Thus, by computing the average exact match between the ground truth and the predicted SQL queries on different components, we can identify limitations of the model and areas where improvements may be necessary.

SQL Hardness criteria: we follow the categorization as described in (Yu et al., 2018a) to understand the performance of the models on SQL queries of varying levels of difficulty. The degree of difficulty here is calculated based on the number of SQL components that the query has. For example queries which have easy level of difficulty have fewer SQL components, selections, and conditions. Whereas queries that have hard level of difficulty may have more SQL keywords such as (GROUP BY, INTERSECT, and ORDER BY).

10 Results

Throughout our experiments, evaluation and inference result generation, we aimed to answer the following research questions:

- **RQ1:**Does varying the inputs to RATSQL transformer by changing pre-trained language model improve the overall accuracy of text2sql generation task?
- **RQ2:** Are we able successfully do a Inference level text2sql generation using prompting based techniques?
- **RQ3:** Can these text to query generation models be able to handle a complex question? Does it help if we break this complex question into simpler subquestions?
- **RQ4:** Can we leaverage the GPT-3.5 capabilities to create an evaluation metric for Text-to-SQL that aligns with human preference?

10.1 Loss and Alignment

Loss values:

We train the baseline and other models from scratch and display the training loss results in 4 for all the iterations having different steps. We see the overall model losses converging at around 6000 iterations and thereby decreasing afterwards 8. To deep dive further, if we look at the losses for the GloVe based model trained

on WikiSQL dataset, the loss starts quite low compared to the GloVe based model trained on Spider dataset, which could be attributed to the complexity of the Spider dataset, with different database and complex relationship across tables. We see a consistent decrease for across train and val datasets as well.
If we look at the models trained on Spider dataset, we see a consistent dip for the overall and train losses. Here Spider + BERT tends to have a slightly higher loss compared to others. And interesting aspect is the slightly upward trend observed with validation datasets across all the models, indicative of possible overfitting of the models. We find RoBERTa, GAP and GRaPPa variants to be performing better, and amongst these RoBERT a seems to have a compartively lesser validation loss, when compared to others as can be seen in 9.

Alignment Loss: To demonstrate the how closely the tokens in the natural language question (utterance) aligns with the schema of the underlying database, we extract the alignment matrix constructed by the RATSQL model (irrespective of the embeddings). The alignment matrix was constructed as part of the RATSQL model (Wang et al., 2020a) consists of a sparse matrix with high values for tokens in the utterance where it is closely related to the tables and columns of the schema. This helps identify 'relevant' columns and tables that needs to be used in the resulting query. Thus, we understand as the model learns, the sparsity increases till the extent we are able to reliably extract 'relevant' structure from the schema.
We demonstrate this using the figure 6 based on training done on RoBERTa but using the same question used in the RATSQL's implementation (Wang et al., 2020a). The intent of doing this was for a quick comparison of how RoBERTa has better alignment representation in comparison to alignment matrix depicted in the original implementation.
The alignment matrix6 has been represented as a correlation heatmap. We make the following observations when looking at the tokens from the natural language:

- First lets look at tables (right side of the x-axis) We see all the values have a very strong relation with table cars_data. Though comparatively lesser, we see a reasonable relation with the table models_list wrt tokens "with 4 cylinders"

- Two columns 'Model' from car_names, and model_list have a correlation with most of the tokens in the sentence except for "4 cylinders", "largest horsepower", which is related with the column "horsepower" from "cars_data".

- Moreover, though a relationship cannot be derived from the alignment matrix, but based on the schema definition, the table car_data has a foreign key mapping with car_names on the column Makeid and with model_list on the column model

And thus the generated sql query "SELECT model_list.Model FROM model_list JOIN car_names ON model_list.Model = car_names.Model AND model_list.Model = car_names.Model JOIN cars_data ON car_names.MakeId = cars_data.Id WHERE cars_data.Cylinders = '4' ORDER BY cars_data.Horsepower Desc LIMIT 1" makes sense.

Additionally, in the Appendix, we note down the alignment matrices generated for other models, and a different question for WikiSQL dataset.

10.2 Results on WikiSQL Dev sets

WikiSQL data set was considered according to several papers as a simpler data set with lot of questions spanning around single database unlike spider. Since, there was huge scope for improvement in case of spider based experiment, we proceeded improvising on the results provided by baseline model for spider data set. However, we trained the baseline model for WikiSQL on our local computer and got the following results with GLoVE embeddings passed as inputs to RAT-SQL.

Models	Steps	LF(%)	EX(%)
RAT-SQL			
Dev	40000	73.6	79.3

Table 1: RAT-SQL accuracy on WikiSQL Dev and Test sets, trained without BERT augmentation or execution-guided decoding (EG). *"LF" = Logical Form Accuracy; "EX" = Execution Accuracy*

10.3 Results on Spider Dev sets

In the following section, we summarize the performance of all the architectures on the Spider Dev sets. The results of RAT-SQL + Glove, RAT-SQL + GraPPa (MLM +SSP), and RAT-SQL + GraPPa (SSP) are taken at 40000 steps, RAT-SQL + RoBERTa at 34100 steps, RAT-SQL + BERT at 40100 steps, and RAT-SQL + GAP at 41000 steps. The Spider dev set comprising 1034 queries is categorized into 248 easy, 446 medium, 174 hard, and 166 extra hard queries.

Logical Form accuracy: Table 2 provides breakdown of the Logical form accuracy grouped by level of difficulty. As expected, we observe that the performance of all the model decreases with increasing difficulty. The drop in accuracy is significant in the extra hard category, highlighting the limitations of the models in handling extremely complex queries. From the results, we also observe that RATSQL + GAP has the highest Logical Form accuracies ($\approx$5-10 % boost over RATSQL + Glove baseline) across all levels of difficulty. GAP, being a graph-based model, emphasizes capturing the structural dependencies and relationships within SQL queries. This focus on query structure may possibly allow RatSQL + GAP to excel in achieving higher logical form accuracy.

Execution accuracy: The execution accuracies across the different architectures are represented in Table 3. We notice that RATSQL + GRAPPA has the highest execution accuracy. GRAPPA, places emphasis on pre-training and capturing important structural properties commonly found in table semantic parsing. By incorporating this, RatSQL + GRAPPA learns to generate queries that not only have the correct logical form but also execute correctly against the underlying database. This could possible lead to a higher execution accuracy.

Component matching accuracy: Table 5 generalizes the component matching accuracies across all the architectures. The SELECT accuracies across the models range from 83.8% (RAT-SQL + Glove) to 92.0% (RAT-SQL + GAP), suggesting that the selection of language representation and model architecture impacts the accuracy of predicting SELECT clauses. RAT-SQL + GAP performs notably better in WHERE accuracy compared to the other variants, indicating the advantage of the graph-based approach in capturing complex conditions and constraints. RAT-SQL + GAP and RAT-SQL + GraPPa (MLM+SSP) demonstrate higher accuracies in these aggregation and sorting components, implying the effectiveness of the graph-based and pre-training approaches in modeling the structure of GROUP BY and ORDER BY clauses. All variants achieve high accuracies in predicting AND/OR, keywords This suggests that the models successfully capture the presence of logical connectors within the queries.

10.4 Results on SParC Test sets

In order to assess the generalizability of our model on an unseen dataset, we sought the SParC dataset which shares similar schemas as our training data Spider. After exploring different options. The SParC dataset contains pairs of utterances and queries in an interactive format From our results in Table 6, it is observed that existing variants of RAT-SQL did not perform well in terms of generalization on the SParC dataset. This can be attributed to the differences in data distribution between the Spider and SParC datasets. An interesting direction to explore in future would be to experiment with prompting-based approaches specifically tailored for the SParC dataset and compare its performance with the existing results.

10.5 Rewritten/Paraphrased Questions

Out of the 1034 samples from ”dev.json” of Spider dataset, we filter out 693 utterances which were evaluated to True for exact match accuracy. Out of the remaining 341 utterances, we select 15 samples to rewrite or paraphrase the utterances and evaluate its result. Based on the outputs, we could draw below observations:

Output 1: We were able to successfully rewrite/-paraphrase 12/15 of these utterances to give an exact match with the gold sql query. Few of the rewriting strategies adopted, which was not ’one-size-fits-all’ and varied based on the input utterance:

- adding tokens semantically similar to columns or tables such as example 2 in 7, where ”degree summary name” had to provided

Models	**Easy**	**Medium**	**Hard**	**Extra Hard**	**All**
RAT-SQL + Glove	79.4	63.0	54.6	40.4	61.9
RAT-SQL + BERT	65.7	47.8	43.7	34.3	49.2
RAT-SQL + RoBERTa	83.1	71.3	53.4	45.8	67.0
RAT-SQL + GAP ↑	91.5	74.8	62.8	44.8	**71.7**
RAT-SQL + GraPPa (MLM +SSP)	85.5	72.6	60.3	48.8	69.8
RAT-SQL + GraPPa (SSP)	86.3	75.3	61.5	49.4	71.5

Table 2: Logical Form accuracies (%) for variants of *RAT-SQL* across different degrees of difficulties.

Models	**Easy**	**Medium**	**Hard**	**Extra Hard**	**All**
RAT-SQL + Glove	48.4	41.5	47.7	37.3	43.5
RAT-SQL + BERT	55.6	44.8	47.7	31.9	45.8
RAT-SQL + RoBERTa	50.8	47.3	50.0	38.0	47.1
RAT-SQL + GAP	49.6	37.4	44.2	16.8	39.0
RAT-SQL + GraPPa (MLM +SSP) ↑	51.2	48.9	52.3	41.6	**48.8**
RAT-SQL + GraPPa (SSP)	51.6	48.4	52.9	39.2	48.5

Table 3: Execution accuracies(%) for variants of *RAT-SQL* across different degrees of difficulties.

Models	**Easy**	**Medium**	**Hard**	**Extra Hard**	**All**
text-davinci-003 (Inference only)	84.7	65.0	54.0	42.8	64.3

Table 4: Execution accuracies(%) of few shot prompting experiment which we conducted on Validation dataset (**Inference only**)

Models	SELECT	WHERE	GROUP BY	ORDER BY	AND/OR	KEYWORDS
RAT-SQL + Glove	83.8	75.1	76.0	73.5	97.1	86.7
RAT-SQL + BERT	85.3	69.1	74.6	77.8	93.8	86.0
RAT-SQL + RoBERTa	91.0	73.6	79.0	85.0	97.8	90.5
RAT-SQL + GAP	92.0	81.1	81.1	85.5	97.8	90.9
RAT-SQL + GraPPa (MLM+SSP)	91.9	72.3	83.6	84.0	98.2	89.8
RAT-SQL + GraPPa (SSP)	91.5	76.3	79.3	81.5	98.2	90.4

Table 5: Component matching accuracies(%) for variants of *RAT-SQL*

- explicit specification to remove columns such as for the utterance ”What is the average and maximum capacity for all stadiums ? without the ’highest’ or ’average’ for a single stadium”
- mention of aggregation strategy such as ”What is the highest rank of losers in all matches.”
- specific mention of order such as ”Find the first name and country code of the player born first” instead of the original utterance ”Find the first name and country code of the oldest player.”
- use of aggregation strategy across columns such as example 1 in 7 were the utterance ”.. average ages of losers and winner ..” had to

Models	Steps	Accuracy(%)
RAT-SQL	40000	32.5
RAT-SQL + BERT	80000	12.78
RAT-SQL + RoBERTa	34200	9.89
RAT-SQL + GAP	41000	17.7
RAT-SQL + GraPPa (MLM +SSP)	40000	11.1
RAT-SQL + GraPPa (SSP)	40000	9.64

Table 6: Accuracies(%) of RAT-SQL with different model combinations on pre-processed SParC used as test set

rephrased to " .. average age of losers and average age of winners ... "

Output 2: We failed to generate exact match predictions, despite multiple rewriting attempts on the taken utterances. Despite the adopting the rewriting strategies mentioned above, the model was unable to generate a prediction that is syntactically similar to the gold sql query. One such example has been provided in 7, where see the resultant query is unable to reorder the columns 'Id' from the table 'car_makers' in the database 'car_1' and also had an additional column 'Fullname' from the same table. While another example had the issue distinguishing the column "Average" from the aggregation method of using an average. While the third error sample included mismatch in the order of columns.

From the outputs, we make the below **observations**:

- we believe more samples for utterance and samples could be provided to help contextualize these cases and map with the syntactic structure of the underlying schema
- it is tough for the decoding algorithm to generate next attribute, unless there is an output template specified to capture the expected output format,
- the input sample would need to be possibly annotated or the embedding should be able to better contextualize and differentiate between special tokens such as "average" between a column or aggregation method.

However, the latter point could be further looked at from the perspective of good sql naming conventions which is generally followed, to avoid naming columns/tables/databases with SQL keywords.

10.6 Analysis of Question/Sub-question

From the accuracy metrics obtained on the test of SParC, we see very low accuracy. As mentioned in the experiment setup, we now introduce questions i.e. the common interaction goal for all the interaction questions we tested and obtained results in the earlier subsection. To demonstrate this, we take few samples to be shown in the table 8.

Outputs

We the first set of interactions where the common interaction goal is a compounded sentence, is being achieved accurately by the model and same with the first interaction sub-question. However, the subsequent interaction is unable to decoded by the model to an accurate sql query. While the last interaction is contextualized based on results from previous interactions.

From the outputs we make the following **observations**:

- The model is able to better perform handling a compounded/complex query, when compared to subquestion based interaction samples in the SParC dataset. This can primarily be attributed to contextualization present in the subquestion interactions in the dataset. This also explains the lower accuracies obtained for models using the SParC dataset.
- Apart from contextualization, the model's inability to generate an accurate sql query, can be due various reasons such as the not being able to correlate tokens with the schema information etc. as mentioned in the subsequent sections of the report. That is to say, compositionality/decompositonality does not seem to have effect for RAT-SQL based models.

Original Question 1: What are the average ages of losers and winners across matches?
Prediction: SELECT Avg(matches.loser_age) FROM matches INTERSECT SELECT matches.loser_age FROM matches
Gold: SELECT avg(loser_age), avg(winner_age) FROM matches
Rewritten Question: What is the average age of losers and average age of winners across matches?
New Prediction: SELECT Avg(matches.loser_age), Avg(matches.winner_age) FROM matches

Original Question 2: How many different degree names are offered?
Prediction: SELECT Count(DISTINCT Courses.course_name) FROM Courses
Gold: SELECT count(DISTINCT degree_summary_name) FROM Degree_Programs
Rewritten Question: What are the different degrees programs offered? Can you give me the count of degree summary names ?
New Prediction: SELECT Count(DISTINCT Degree_Programs.degree_summary_name) FROM Degree_Programs

Original Question 3: What is the full name of each car maker, along with its id and how many models it produces?
Prediction: SELECT car_makers.Maker, car_makers.FullName, car_makers.Id, Count(*) FROM car_makers
JOIN model_list ON car_makers.Id = model_list.Maker GROUP BY car_makers.Id
Gold: SELECT T1.FullName , T1.Id, count(*) FROM CAR_MAKERS AS T1 JOIN MODEL_LIST AS T2 ON T1.Id = T2.Maker
GROUP BY T1.Id;
Rewritten Question: Give me only the id of each car maker and the count of the models along with its full name
New Prediction: SELECT car_makers.Id, car_makers.FullName, Count(*) FROM car_makers JOIN model_list
ON car_makers.Id = model_list.Maker GROUP BY car_makers.Id, car_makers.FullName

Table 7: Rewritten, Paraphrased Question Results. These results have been generated on RoBERTa checkpointed model at 34100.

C_1 **:** What are the id, name and membership level of visitors who have spent the largest amount of money in total in all museum tickets?
P_1 **:** SELECT visitor.ID, visitor.Name, visitor.Level_of_membership FROM visitor JOIN visit ON visitor.ID = visit.visitor_ID
GROUP BY visitor.ID ORDER BY Sum(visit.Total_spent) Desc LIMIT 1
Correctly generated query with result ('3', 'Arjen Robben', 1)
Q_1 **:** What is the total spent on all visits?
S_1 **:** SELECT sum(Total_spent) FROM visit
Correctly generated query with result (980.96)
Q_2 **:** Find the name of the visitor who has spent the most money for his or her visits.
S_2 **:** SELECT visitor.Level_of_membership FROM visitor JOIN visit ON visitor.ID = visit.visitor_ID GROUP BY visit.visitor_ID
ORDER BY Sum(visit.Num_of_Ticket) Desc LIMIT 1
Incorrectly generated query
Q_3 **:** What are his id and membership level?
S_3 **:** SELECT museum.Num_of_Staff, visitor.Level_of_membership FROM museum JOIN visitor JOIN visit
ON museum.Museum_ID = visit.Museum_ID AND visit.visitor_ID = visitor.ID WHERE museum.Name = 'terminal'
Contextualization using 'his' and incorrectly generated query

Table 8: Question/Sub-question interaction sample from SParC dataset and query results P_1, S_1, S_2, S_3 generated using RoBERTa. Here the notation Q_m denotes subquestions asked to complete the interaction goal C_n, and P_n denotes the corresponding sql query prediction, whereas S_m denotes the sql queries generated for subquestions Q_m. This extends the structure mentioned in (Yu et al., 2019) with slight modifications. Here 'terminal' denotes a token assinged by RAT-SQL as a placeholder for the value.

11 Ablation Studies

11.1 RAT relation linking

Natural language queries can be ambiguous, especially when they involve similar or overlapping terms. Schema linking helps disambiguate these terms by mapping them to specific columns or tables in the database schema. Schema linking allows the model to adapt to diverse databases by identifying the relevant elements in the schema during inference. In this section, we study the significance of schema linking of RATSQL + BERT architecture.

Table 9 contrasts the performance of RATSQL + BERT with and without Schema linking. For the latter variant, we remove the schema linking component and train the RATSQL + BERT architecture over 40000 epochs on the Spider train set. From the results on Spider dev set, we can observe that as we remove the schema linking component, the execution accuracy of the model drops by $\approx 2\%$ hinting that schema linking is neccessary in helping the model understand relation between the query and the database.

Models	Accuracy (%)
RAT-SQL+BERT + value linking	
w/o schema linking relations	43.7
w schema linking relations	45.8

Table 9: Execution accuracies of RAT-SQL and RAT-SQL + BERT ablations on the Spider dev set.

11.2 Oracle Sketch and Oracle Column

We conduct experiments to evaluate the performance (Table 10) of the different RATSQL variants on 'oracle sketch' and 'oracle column' techniques for the spider dataset. From our analysis, we identify that across all variants, using the 'oracle columns' technique, the accuracy of the models are $\approx 77\%$. This means that the model correctly answers $\approx 77\%$ of the questions when it is forced to emit the correct column or table at the terminal nodes during decoding.

Based on this result, it can be inferred that the model's accuracy significantly improves when it is provided with the correct column or table information. However, even with the 'oracle column' technique, the model still gets some questions wrong, indicating that there are other aspects or challenges that contribute to the inaccuracies.

Using the 'oracle sketch' technique, the accuracy of the models are $\approx 98\%$. These results signify that the model correctly answers $\approx 98\%$ of the questions when it is forced to choose the correct production at every grammar nonterminal, resulting in the exact match of the SQL sketch with the ground truth. The high accuracy achieved with the "oracle sketch" technique suggests that the model's capability to generate the correct SQL sketch is a crucial factor in accurately answering natural language queries.

12 Error Analysis

Classification of Errors The different classes of error suggested here is same as work done in (Lee et al., 2021). We parsed into the JSON given as output by the model to extract details regarding each clause and we did the below classification. Incorrect constraints are usually the cases where the 'WHERE', 'HAVING' and 'ORDER BY' clause in the JSON is not the same between and gold and predicted but everything except them remains the same. Missing constraints are when there is a constraint in the gold query but the same constraint is missing in case of predicted query. For this, we compare the two JSONs which are exactly the same but except for three clauses 'WHERE', 'ORDER BY' and 'HAVING'. Entity column mismatch are those cases where the 'WHERE' clause differs between gold and predicted query because of the column referenced. Major structural changes between predicted and gold query can be classified as understanding error. There were few instances we could not capture in any of these cases due to the mismatch in the order in which constraints were appended between gold and predicted queries. Since the structure was too complex, we could not sort the tuples and work on such cases. But we were able to classify good percentage of queries (approximately 30%) into one of the above categories. An example for each of the below category is shown in Table 13 for the baseline models and the model proposed by us.

- **Incorrect Final Column** There are two types of final column errors: incorrect predictions, and ambiguous predictions. In incorrect predictions, the final column type is obviously incorrect with respect to the utterance. In ambiguous predictions, the final column is a reasonable prediction with respect to the utterance, but due to the dataset conventions, is incorrect. Both of these errors are classified as incorrect final column errors as demonstrated in Table 13.

- **Missing Constraint and Incorrect Constraint** We consider two types of errors related to incorrectly implementing the utterance's intent in SQL: missing and incorrect constraints. Missing constraints involve ignoring a constraint mentioned in the utterance, such as the constraint about government form is ignored in case of SPIDER+GLOVE. Incorrect constraints are when we have cases where a query is sorted in ascending order instead of descending order or when the WHERE clause or HAVING

Models	Steps	Accuracy (%)
RAT-SQL + RoBERTa	34100	
+ Oracle Columns		77.8
+ Oracle Sketch		98.2
RAT-SQL + GAP	41000	
+ Oracle Columns		76.5
+ Oracle Sketch		97.7
RAT-SQL + GraPPa (MLM +SSP)	40000	
+ Oracle Columns		76.8
+ Oracle Sketch		98.2
RAT-SQL + GraPPa (SSP)	40000	
+ Oracle Columns		78.9
+ Oracle Sketch		98.2

Table 10: Accuracy (exact match %) on the Spider development set given an oracle providing correct columns and tables ("Oracle columns") or the AST sketch structure ("Oracle sketch").

clause is using wrong set of columns to get the output (this is also called entity column matching).

- **Entity column matching** We consider column matching errors under this category. Entity column errors corresponds to those instances where we compare an entity that we ask the question about to the wrong column in the database. For example in Table 13, we have to identify those ships that end up being captured. But instead of looking for the string Captured in the disposition of ship column, SPIDER+GLOVE, for example looks in to column named as type of ship.
- **Understanding Error** We classify few examples where the query predicted by the model is structurally different from the gold query as understanding errors. This could happen due to too less specifications in the question asked or it could be because of the complex structure of the question. In the Table 13, we see that instead of comparing Ford motor company against full name, the roberta based output compares it against maker. This makes the output of the query wrong. Prompting based techniques which generally succeeds in cases where the details are specified fails here due to lack of understanding of the schema.

Issues

- There are few instances where we couldn't classify the errors into any of the above classes for which we had to manually do a eye-balling. There were 341 mis-predicted queries in case of RATSQL+Roberta and 396 in case of RATSQL+GLOVE. But the error classification code written us by was able to classify roughly 80 queries into one of the above buckets. We used the predicted JSON for performing this task.
- There were also instances where the execution of the query gave the same results but the exact accuracy was False(due to structural mismatch) but execution gave out right results, but due to the order in which the columns were retrieved or due to the order in which results were stored between gold and predicted queries, we couldn't filter out these type of queries because execution accuracy gave out false. One such example is, SELECT CountryName FROM countries EXCEPT SELECT T1.CountryName FROM countries AS T1 JOIN CAR_MAKERS AS T2 ON T1.countryId = T2.Country is the gold query for which predicted query(RATSQL+Roberta) looks like this, SELECT countries.CountryName FROM countries WHERE countries.CountryId NOT IN (SELECT car_makers.Country FROM car_makers). These two queries outputs [('australia',), ('brazil',), ('egypt',), ('mexico',), ('new zealand',), ('nigeria',), ('russia',)] and [('russia',), ('nigeria',), ('australia',), ('new zealand',), ('egypt',), ('mexico',), ('brazil',)] respectively. But the

order of these two lists are different.

- Another issue is that, all implementations except prompting based experiment decodes 'terminal' for any entity asked in the question as stated in Table 11 and 12.
- As we look at error analysis using RAT-SQL+BERT, we had flagged a parameter called as include_literal as true in config file. This is the reason why, we see a lot of invalid values for BERT related experiments instead of 'terminal'.
- Due to the above stated issues, there were few instances wherein we had one mis predicted query classified as more than one type of error and at times not being classified into any of these. The discussion below are mostly about the queries that were solely caused due to one of the above reasons.

Error Analysis Results of Each Experiment For all experiments relating to spider dataset with RATSQL, we do the analysis in terms of exact match accuracy at query level (of validation dataset) and deep dive into the error classes when exact match accuracy is False. But, there were instances where exact match accuracy were wrong for few cases but the execution accuracy was true. This is because any SQL query can be written in multiple ways to extract the same output (semantically equivalent queries). We see below the error analysis done for spider and wikisql dataset using RATSQL transformer. In case of spider, we start with using GloVe (Global Vectors for Word Representation) passed as input to RAT-SQL transformer and changing the word embedding to a contextualized embeddings using pre-trained language model like BERT,RoBERTa etc.., Pre-trained language models like RoBERTa and BERT provide contextual embeddings, meaning that each token's embedding is generated based on the context of the entire sentence. This allows for a more nuanced understanding of the sentence's meaning, which can be especially useful for natural language processing tasks like RatSQL, which require a deep understanding of language. Pre-trained language models have a much larger vocabulary than GLOVE, allowing for a more accurate representation of rare or domain-specific words. This can be particularly important for tasks like RatSQL, which may involve technical vocabulary. We observe the number of errors in each of the above mentioned class and understand the reasons behind it. Example queries along with error class for each of the experiments (except GRAPPA and GAP) stated below is provided in Table 13.There are several advantages of using pre-trained language models for embedding over GLOVE in RatSQL:

- **RATSQL for Spider** Result metrics of spider validation dataset which consisted of 1034 questions were analyzed for GLoVe embedding based as input to RATSQL transformer. It is seen that out of 394 mispredicted queries (according to exact match accuracy at query level), we were able to categorize 333 ones into category that execution result does not match between gold and predicted queries. Out of 333 queries, we had about 57 queries which were incorrect because it did not retrieve the right column as output of the query. We had 16 of them which had a missing constraint which it was unable read from the question given as input. For example, the natural language question What is the average, minimum, and maximum age for all French singers? had a a gold query defined as SELECT avg(age) , min(age) , max(age) FROM singer WHERE country = 'France'. But the predicted query missed the constraint that the country needs to be France. The predicted query corresponding to this question was SELECT Avg(singer.Age), Min(singer.Age), Max(singer.Age) FROM singer. We also had 21 errors which corresponds to the class of incorrect constraint and 14 of them correspond to entity column mismatch.
- **RATSQL + BERT for Spider** Result metrics of spider validation dataset which consisted of 1034 questions were analyzed for GLoVe embedding based as input to RATSQL transformer. It is seen that out of 501 mispredicted queries (according to exact match accuracy at query level), we were able to categorize 451 ones into category that execution result does not match between gold and predicted queries. Out of 451 queries, we had about 37 queries which were incorrect because it did not retrieve the right column as output of the query. We had 17 of them which had a missing constraint which it was unable read from

Question: Show all paragraph ids and texts for the document with name 'Welcome to NY'.
Gold: SELECT T1.paragraph_id , T1.paragraph_text FROM Paragraphs AS T1 JOIN Documents AS T2 ON T1.document_id = T2.document_id WHERE T2.Document_Name = 'Welcome to NY'

Prediction (RATSQL+GLoVE): SELECT Paragraphs.Paragraph_ID, Documents.Document_Name FROM Documents JOIN Paragraphs ON Documents.Document_ID = Paragraphs.Document_ID WHERE Documents.Document_Name = 'terminal'
Prediction (RATSQL+BERT): SELECT Paragraphs.Paragraph_ID, Paragraphs.Paragraph_Text FROM Documents JOIN Paragraphs ON Documents.Document_ID = Paragraphs.Document_ID WHERE Documents.Document_Name = <UNK>
Prediction (RATSQL+RoBERTa): SELECT Paragraphs.Paragraph_ID, Paragraphs.Paragraph_Text FROM Documents JOIN Paragraphs ON Documents.Document_ID = Paragraphs.Document_ID WHERE Documents.Document_Name = 'terminal'
Prediction (RATSQL+GRAPPA(SSP)): SELECT Paragraphs.Paragraph_ID, Paragraphs.Paragraph_Text FROM Documents JOIN Paragraphs ON Documents.Document_ID = Paragraphs.Document_ID WHERE Documents.Document_Name = 'terminal'
Prediction (RATSQL+GRAPPA(SSP+MLM)): SELECT Paragraphs.Paragraph_ID, Paragraphs.Paragraph_Text FROM Documents JOIN Paragraphs ON Documents.Document_ID = Paragraphs.Document_ID WHERE Documents.Document_Name = 'terminal'
Prediction (RATSQL+GAP): SELECT Paragraphs.Paragraph_ID, Paragraphs.Paragraph_Text FROM Documents JOIN Paragraphs WHERE Documents.Document_Name = 'terminal' - Missing joint constraint
Prediction (Prompting(OpenAI-text-davinci-003)): SELECT Paragraph_ID, Paragraph_Text FROM Paragraphs JOIN Documents ON Paragraphs.Document_ID = Documents.Document_ID WHERE Documents.Document_Name = 'Welcome to NY'

Table 11: Comparison of Predictions done across Experiments - **SQL Difficulty: Medium**

Question: Find the major and age of students who do not have a cat pet.
Gold: SELECT major , age FROM student WHERE stuid NOT IN (SELECT T1.stuid FROM student AS T1 JOIN has_pet AS T2 ON T1.stuid = T2.stuid JOIN pets AS T3 ON T3.petid = T2.petid WHERE T3.pettype = 'cat')

Prediction (RATSQL+GLoVe): SELECT Student.Major, Student.Age FROM Student WHERE Student.StuID NOT IN (SELECT Has_Pet.StuID FROM Has_Pet JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = 'terminal')
Prediction (RATSQL+BERT): SELECT Student.Major, Student.Age FROM Student WHERE Student.StuID NOT IN (SELECT Has_Pet.StuID FROM Has_Pet JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = "TV Lounge")
Prediction (RATSQL+RoBERTa): SELECT Student.Major, Student.Age FROM Student WHERE Student.StuID NOT IN (SELECT Has_Pet.StuID FROM Student JOIN Has_Pet ON Student.StuID = Has_Pet.StuID AND Student.StuID = Has_Pet.StuID JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = 'terminal')
Prediction (RATSQL+GRAPPA(SSP)): SELECT Student.Major, Student.Age FROM Student EXCEPT SELECT Student.Major, Student.Age FROM Student JOIN Has_Pet ON Student.StuID = Has_Pet.StuID AND Student.StuID = Has_Pet.StuID JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = 'terminal'
Prediction (RATSQL+GRAPPA(SSP+MLM)): SELECT Student.Major, Student.Age FROM Student WHERE Student.StuID NOT IN (SELECT Has_Pet.StuID FROM Has_Pet JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = 'terminal')
Prediction (RATSQL+GAP): SELECT Student.Major, Student.Age FROM Student WHERE Student.StuID NOT IN (SELECT Has_Pet.StuID FROM Has_Pet JOIN Pets ON Has_Pet.PetID = Pets.PetID WHERE Pets.PetType = 'terminal')
Prediction (Prompting(OpenAI-text-davinci-003)): SELECT Major, Age FROM Student WHERE StuID NOT IN (SELECT StuID FROM Has_Pet WHERE PetID IN (SELECT PetID FROM Pets WHERE PetType = 'cat'))

Table 12: Comparison of Predictions done across Experiments - **SQL Difficulty: Extra Hard**

the question given as input. We also had 18 errors which corresponds to the class of incorrect constraint and 5 of them correspond to entity column mismatch.

- **RATSQL + RoBERTa for Spider** Result metrics of spider validation dataset which consisted of 1034 questions were analyzed as we use the pretrained language model RoBERTa to generate contextualized embedding given as input to RATSQL transformer. It is seen that out of 341 mispredicted queries (according to exact match accuracy at query level), we were able to categorize 275 ones into category that execution result does not match between gold and predicted queries.

Out of 275 queries, we had about 32 queries which were incorrect because it did not retrieve the right column as output of the query. We had 8 of them which had a missing constraint which it was unable read from the question given as input. We also had 16 errors which corresponds to the class of incorrect constraint and 25 of them corresponding to entity column mismatch.

- **RATSQL + GraPPa (MLM + SSP) for Spider**

  Result metrics of spider validation dataset which consisted of 1034 questions were analyzed on the RATSQL + GraPPa (MLM + SSP) architecture. It is seen that out of the 310 mispredicted queries (according to exact match accuracy at query level), we were able to categorize 240 ones into category that execution result does not match between gold and predicted queries. Out of 240 queries, we had about 28 queries which were incorrect because it did not retrieve the right column as output of the query. We had 3 of them which had a missing constraint which it was unable read from the question given as input. We also had 15 errors which corresponds to the class of incorrect constraint and 15 of them corresponding to entity column mismatch.

- **RATSQL + GraPPa (SSP) for Spider**

  Result metrics of spider validation dataset which consisted of 1034 questions were analyzed on the RATSQL + GraPPa (SSP) architecture. It is seen that out of the 312 mispredicted queries (according to exact match accuracy at query level), we were able to categorize 268 ones into category that execution result does not match between gold and predicted queries. Out of 268 queries, we had about 26 queries which were incorrect because it did not retrieve the right column as output of the query. We had 10 of them which had a missing constraint which it was unable read from the question given as input. We also had 18 errors which corresponds to the class of incorrect constraint and 17 of them corresponding to entity column mismatch.

- **Few shot and Zero Shot prompting for Spider** As we look into the results generated by few shot prompting, we were able to classify 87 out of 369 mispredicted queries into on of the above errors listed. It is seen that there is a huge number of incorrect final column error which is about 80%. This means that, inference only prompting were not enough for the API to learn the instruction and capture all the columns that it needs to output from the question given. But, we weren't able to test to improved prompts as we ran out of openai credits. The accuracy mentioned in table 4 is based on the first cut experiment that we tried with prompting. There are also instances where, because a value linking is done, due to case sensitive issues, few queries fail. For example, Gold query is SELECT avg(T2.edispl) FROM CAR_NAMES AS T1 JOIN CARS_DATA AS T2 ON T1.MakeId = T2.Id WHERE T1.Model = 'volvo'. But inference using text-davinci-003 engine generates SELECT AVG(Edispl) FROM cars_data AS T1 JOIN model_list AS T2 ON T1.ModelId = T2.ModelId WHERE T2.Model = 'Volvo' as predicted query. Over here, because 'Volvo' and 'volvo' are not the same with respect to records in database, we have a 0 in case of accuracy for this query. But, prompting do help the query understand the questions well when well-specified as we see it Table 11 and 12. Overall, there is a huge scope for improvement in this area either by improving the prompts that were constructed or by fine tuning the API to output better results.

- **Human-aligned evaluation with GPT-3.5 and GPT-4**
  A random sampling of 1% of all questions in the Spider dataset benchmark was performed as stated in 14. As a result, there were 86 questions from training examples and 10 questions from development examples.
  The generated SQL queries from these 96 questions were evaluated, and the overall accuracy for GPT-3.5 ***code-davinci-002*** is **79.17%**, ***text-davinci-003*** is **88.54%**, ***gpt-3.5-turbo*** is **85.42%**, and ***gpt-4*** is **89.58%**. As expected, GPT-4 outperformed the other models, which aligns with our practical experience of these models' relative performance. However, the execution and match accuracies of GPT models do not align with the practical experience as mentioned in 7. Using

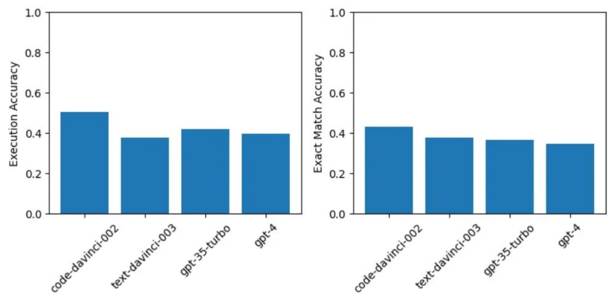


Figure 7: Execution and exact match accuracies of GPT models on 96 questions sampled from Spider benchmark

these metrics as the primary measurement of Text-to-SQL performance, one would think gpt-4 is less powerful than code-davinci-002. These results clearly do not match our practical experience with these models.

- **RATSQL for WikiSQL** We did not do a comprehnesive error analysis for wikisql as we just ran the baseline model and reported the accuracy metric. However, we did go through the evaluation file and found that there were several instances were the model gave out semantically equivalent queries (execution accuracy came out as true and logical form accuracy was false). Most of our focus of the error analysis was on implementations done on spider dataset. However, we wanted to explore the prompting based apprach for wikisql given the fact the dataset is not complex as we see in the EDA performed for wikiSQL. However, due to resource constraint we couldn't perform this. Key implications of the human evaluation experiment:
  - It was found that the existing metrics, namely execution accuracy and exact match accuracy, are misaligned with human preference. As a result, they should not be relied upon as the primary way to evaluate Text-to-SQL models.
  - While human evaluation remains the gold standard, large language models (LLMs) such as GPT-3.5 and above can accurately approximate human preference. This is a potential game changer, because they allow us to evaluate Text-to-SQL models and estimate the expected human evaluation result. This is especially useful when human evaluation is not feasible due to budget limitation.
  - The use of instruction-tuned LLMs for evaluation could be generalized to many other tasks. This approach is especially useful for tasks whose output consists of simple statements that LLMs can directly reason over. And there's a research paper on this very topic called "GPTScore" (Fu et al., 2023), which trains GPT-3.5 to evaluate NLP tasks using in-context learning.

13 Contributions of Group Members

All the members in the group contributed equally for making the report.

- Prachi Jain: Worked on setting up experiments RAT-SQL, RAT-SQL + BERT and RAT-SQL + GAP based models for the Spider dataset. Contributed to codebase to make GAP based implementation. Trained RAT-SQL variant, Spider + BERT variants for rat-based relations ablation study. Worked on setting up Oracle based evaluation for column and sketch for RAT-SQL + GAP, RAT-SQL + BERT and RAT-SQL . Performed testing of RAT-SQL, RAT-SQL + BERT and RAT-SQL + GAP on SParC dataset. Designed and performed Human-Aligned Text-to-SQL Evaluation for Spider Dataset using LlamaIndex Open Source Project and extended it to GPT3.5 (code-davinci-002, text-davinci-003, turbo: APIs released by OpenAI) and GPT-4 models in zero - shot setting, Debugging and solving error issues

- Vijayalakshmi Vasudevan: Trained WikiSQL+Glove model and obtained the baseline logical form and execution accuracy, Contributed to codebase to make RoBERTa based implementation, worked on setting up experiment in google colab (using stanza instead for coreNLP for RAT-SQL+Roberta implementation), worked on building and training Roberta based model for spider dataset, worked on generating prompts for spider dataset, Worked on few shot and zero shot prompting using text-davinci-003 engine (GPT-3) of the API released by openai, Worked on building a code to identify mis-predicted errors and to classify error types, Done a comprehensive error analysis and worked on generating examples for each error type under all experiments conducted, Worked on annotations of errors for all four experiments (RATSQL+BERT,GLOVE,Prompting and Roberta), worked on debugging and solving error issues. Worked on performing oracle sketch and oracle column experiment for Roberta based implementation, worked on computing test accuracy for sparc dataset for RATSQL+Roberta implementation. Contributed to evaluation metrics file for finding evaluation metrics for prompting based techniques.

- Nandhinee Periyakaruppan: Trained RAT-SQL + GraPPa (MLM + SSP), RAT-SQL + GraPPa (SSP) models for the Spider dataset, Trained RAT-SQL + BERT variant for schema linking ablation study and analyzed the test results, Contributed to codebase for developing an evaluation pipeline comprising three text-to-sql metrics, Performed exploratory data analysis on Spider dataset, Tested RAT-SQL + GraPPa (MLM + SSP) and RAT-SQL + GraPPa (SSP) models on SParC dataset, Analyzed extensively the evaluation results across six different variants of RAT-SQL, Performed error analysis on Grappa models classifying the errors into different error types, Performed oracle sketch and oracle column experiments for Grappa models, Debugging and solving error issues

- Shebin Scaria: Infra setup code and helper files for different model codebase setups, base inference setup files for running experiments, contribution to ratsql code base to extract alignment matrix comparisons, preprocessing for SParC dataset, error analysis and annotations. - debugging solving error issues through different parts of codebase. Ran experiments for rewritten/paraphrased utterances, and on interaction based experiments using SParC dataset.

14 Conclusion

We faced a lot of difficulties with respect to compute resources for conducting all of the above stated experiments. Error analysis was a bit challenging as we had multiple experiments performed. We would like to work on the prompting based experiments and fine tuning APIs released by large language model to improve the accuracy. We would also like to work on the execution accuracy module as part of evaluation metric as the current structure has few false negatives due to order mismatch of the columns or the query output. We would like to explore fine tuning sequence to sequence model like T5 for text2sql task.

15 AI Disclosure

- Did you use any AI assistance to complete this project report? If so, please also specify what AI you used.
  - No

If you answered yes to the above question, please complete the following as well:

- If you used a large language model to assist you, please paste *all* of the prompts that you used below. Add a separate bullet for each prompt, and specify which part of the proposal is associated with which prompt.
  - your response here
- **Free response:** For each section or paragraph for which you used assistance, describe your overall experience with the AI. How helpful was it? Did it just directly give you a good output, or did you have to edit it? Was its output ever obviously wrong or irrelevant? Did you use it to generate new text, check your own ideas, or rewrite text?
  - your response here

References

Bogin, B., Gardner, M., and Berant, J. (2019a). Global reasoning over database structures for text-to-sql parsing. *CoRR*, abs/1908.11214.

Bogin, B., Gardner, M., and Berant, J. (2019b). Representing schema structure with graph neural networks for text-to-sql parsing. *CoRR*, abs/1905.06241.

Catherine Finegan-Dollak, Jonathan K. Kummerfeld, L. Z. K. R. S. S. R. Z. D. R. (2018). Improving text-to-sql evaluation methodology. *ACL*, pages 351–360.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional transformers for language understanding.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H.-W. (2019). Unified language model pre-training for natural language understanding and generation.

Fu, J., Ng, S.-K., Jiang, Z., and Liu, P. (2023). Gptscore: Evaluate as you desire.

Guo, J., Zhan, Z., Gao, Y., Xiao, Y., Lou, J.-G., Liu, T., and Zhang, D. (2019). Towards complex text-to-sql in cross-domain database with intermediate representation.

Herzig, J., Nowak, P. K., Müller, T., Piccinno, F., and Eisenschlos, J. M. (2020). TAPAS: weakly supervised table parsing via pre-training. *CoRR*, abs/2004.02349.

Katsogiannis-Meimarakis, G., K. G. (2023). A survey on deep learning approaches for text-to-sql. *The VLDB Journal*.

Lee, C.-H., Polozov, O., and Richardson, M. (2021). KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension.

Li, J., Chen, Z., Chen, L., Zhu, Z., Li, H., Cao, R., and Yu, K. (2023). Dir: A large-scale dialogue rewrite dataset for cross-domain conversational text-to-sql. *Applied Sciences*, 13(4).

Li, Y., Yang, H., and Jagadish, H. V. (2006). Constructing a generic natural language interface for an xml database. In *Proceedings of the 10th International Conference on Advances in Database Technology*, EDBT'06, page 737–754, Berlin, Heidelberg. Springer-Verlag.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019a). Roberta: A robustly optimized bert pretraining approach.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019b). Roberta: A robustly optimized bert pretraining approach. *ArXiv*, abs/1907.11692.

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019c). Roberta: A robustly optimized bert pretraining approach.

Peng Shi, Patrick Ng, Z. W. H. Z. A. H. L. J. W. C. N. d. S. B. X. (2021). Learning contextual representations for semantic parsing with generation-augmented pre-training. *AAAI Technical Track on Speech and Natural Language Processing II*, 35(15):13806–13814.

Popescu, A.-M., Armanasu, A., Etzioni, O., Ko, D., and Yates, A. (2004). Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 141–147, Geneva, Switzerland. COLING.

Popescu, A.-M., Etzioni, O., and Kautz, H. (2003). Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces*, IUI '03, page 149–157, New York, NY, USA. Association for Computing Machinery.

Press, O., Zhang, M., Min, S., Schmidt, L., Smith, N. A., and Lewis, M. (2022). Measuring and narrowing the compositionality gap in language models.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer.

Rajkumar, N., Li, R., and Bahdanau, D. (2022). Evaluating the text-to-sql capabilities of large language models.

Tao Yu, Chien-Sheng Wu, X. V. L. B. W. Y. C. T. X. Y. D. R. R. S. C. X. (2021). Grappa: Grammar-augmented pretraining for table semantic parsing. *ICLR*.

Wang, B., Shin, R., Liu, X., Polozov, O., and Richardson, M. (2020a). 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*, pages 7567–7578, Online. Association for Computational Linguistics.

Wang, R., Tang, D., Duan, N., Wei, Z., Huang, X., ji, J., Cao, G., Jiang, D., and Zhou, M. (2020b). K-adapter: Infusing knowledge into pre-trained models with adapters.

Warren, D. H. and Pereira, F. C. (1982). An efficient easily adaptable system for interpreting natural language queries. *American Journal of Computational Linguistics*, 8(3-4):110–122.

Ye, X., Yavuz, S., Hashimoto, K., Zhou, Y., and Xiong, C. (2021). Rng-kbqa: Generation augmented iterative ranking for knowledge base question answering. *CoRR*, abs/2109.08678.

Yin, P., Neubig, G., Yih, W., and Riedel, S. (2020). Tabert: Pretraining for joint understanding of textual and tabular data. *CoRR*, abs/2005.08314.

Yu, T., Yasunaga, M., Yang, K., Zhang, R., Wang, D., Li, Z., and Radev, D. (2018a). SyntaxSQLNet: Syntax tree networks for complex and cross-domain text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1653–1663, Brussels, Belgium. Association for Computational Linguistics.

Yu, T., Zhang, R., Yang, K., Yasunaga, M., Wang, D., Li, Z., Ma, J., Li, I., Yao, Q., Roman, S., Zhang, Z., and Radev, D. (2018b). Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

Yu, T., Zhang, R., Yasunaga, M., Tan, Y. C., Lin, X. V., Li, S., Er, H., Li, I., Pang, B., Chen, T., Ji, E., Dixit, S., Proctor, D., Shim, S., Kraft, J., Zhang, V., Xiong, C., Socher, R., and Radev, D. (2019). Sparc: Cross-domain semantic parsing in context.

Zhong, V., Xiong, C., and Socher, R. (2017). Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

Pain Point: Incorrect Constraint
Question: What is the name of the winner with the most rank points who participated in the Australian Open tournament?
Prediction (Spider+GLoVe): SELECT matches.winner_name FROM matches WHERE matches.tourney_name = 'terminal' ORDER BY matches.winner_age Desc LIMIT 1
Prediction (Spider+BERT):SELECT matches.winner_name FROM matches WHERE matches.tourney_name = "Orton" ORDER BY matches.winner_rank_points Desc LIMIT 1
Prediction (Spider+RoBERTa):SELECT matches.winner_name FROM matches WHERE matches.tourney_name = 'terminal' ORDER BY matches.winner_rank_points Desc LIMIT 1
Prediction (Spider+Prompting(OpenAI - text-davinci-003): SELECT winner_name FROM matches JOIN rankings ON matches.winner_id = rankings.player_id WHERE tourney_name = 'Australian Open' ORDER BY ranking_points DESC LIMIT 1

Gold: SELECT winner_name FROM matches WHERE tourney_name = 'Australian Open' ORDER BY winner_rank_points DESC LIMIT 1
Error: In this case the constraint used by GLOVE is incorrect as it sorts age by descending order instead of ranking points. This can be termed as incorrect constraint.

Pain Point: Missing Constraint
Question: Give the average life expectancy for countries in Africa which are republics
Prediction (Spider+GLoVe): SELECT Avg(country.LifeExpectancy) FROM country WHERE country.Continent = 'terminal'
Prediction (Spider+BERT):SELECT Avg(country.LifeExpectancy) FROM country WHERE country.Continent = <UNK>
Prediction (Spider+RoBERTa): SELECT Avg(country.LifeExpectancy) FROM country WHERE country.Region = 'terminal' AND country.Continent = 'terminal'
Prediction (Spider+Prompting(OpenAI - text-davinci-003): SELECT avg(LifeExpectancy) FROM country WHERE Continent = 'Africa' AND GovernmentForm = 'Republic'

Gold: SELECT avg(LifeExpectancy) FROM country WHERE Continent = "Africa" AND GovernmentForm = "Republic"
Error: Missing constraint error refers to cases where the predicted query misses out on the constraints mentioned in the question.

Pain Point: Incorrect Final Column
Question: Which professionals live in a city containing the substring 'West'? List his or her role,street,city and state.
Prediction (Spider+GLoVe): SELECT Professionals.role_code, Professionals.street, Professionals.city FROM Professionals WHERE Professionals.city = 'terminal' OR Professionals.city = 'terminal'
Prediction (Spider+BERT):SELECT Professionals.role_code, Professionals.street, Professionals.city, Professionals.state FROM Professionals WHERE Professionals.city LIKE <UNK>
Prediction (Spider+RoBERTa): SELECT Professionals.first_name, Professionals.street, Professionals.city, Professionals.state FROM Owners JOIN Dogs ON Owners.owner_id = Dogs.owner_id JOIN Professionals JOIN Treatments ON Dogs.dog_id = Treatments.dog_id AND Treatments.professional_id = Professionals.professional_id WHERE Owners.city LIKE 'terminal'
Prediction (Spider+Prompting OpenAI - text-davinci-003: SELECT role_code, street, city, state FROM Professionals WHERE city LIKE '%West%'

Gold: SELECT role_code , street , city , state FROM professionals WHERE city LIKE '%West%'
Error: This is because number of columns that needs to be retrieved here is not properly done due to too many asks in the question. But the prompting based technique is working out here as the question clearly specifies the details it needs.

Pain Point: **Entity Column Mismatch**
Question: How many ships ended up being 'Captured'?
Prediction (Spider+GLoVe): SELECT Count(*) FROM ship WHERE ship.ship_type = 'terminal'
Prediction (Spider+BERT):SELECT Count(*) FROM ship WHERE ship.disposition_of_ship = <UNK>
Prediction (Spider+RoBERTa): SELECT Count(*) FROM battle JOIN ship ON battle.id = ship.lost_in_battle WHERE battle.result = 'terminal'
Prediction (Spider+Prompting OpenAI - text-davinci-003): SELECT COUNT(*) FROM ship WHERE disposition_of_ship = 'Captured'

Gold: SELECT count(*) FROM ship WHERE disposition_of_ship = 'Captured'
Error: In this case the entity Captured (which RATSQL term as terminal) is mapped wrongly to the column named lost in battle instead of disposition of ship. In case of RoBERTa, the query is mapped to wrong table in the same database

---

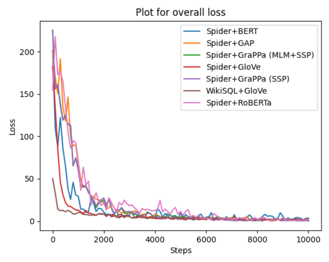
Pain Point: **Understanding Error - Failing to understand the question to compose complex query**
Question: What are the different models that are lighter than 3500 but were not built by the Ford Motor Company?
Prediction (Spider+GLoVe):SELECT DISTINCT model_list.Model FROM model_list GROUP BY model_list.ModelId HAVING Count(*) > 'terminal' EXCEPT SELECT DISTINCT model_list.Model FROM model_list WHERE model_list.Model = 'terminal'
Prediction (Spider+BERT):SELECT DISTINCT model_list.Model FROM car_makers JOIN model_list ON car_makers.Id = model_list.Maker AND car_makers.Id = model_list.Maker JOIN car_names ON model_list.Model = car_names.Model JOIN cars_data ON car_names.MakeId = cars_data.Id WHERE cars_data.Weight < 0 EXCEPT SELECT DISTINCT model_list.Model FROM car_makers JOIN model_list ON car_makers.Id = model_list.Maker WHERE car_makers.FullName = <UNK>
Prediction (Spider+RoBERTa): SELECT DISTINCT model_list.Model FROM model_list JOIN car_names ON model_list.Model = car_names.Model JOIN cars_data ON car_names.MakeId = cars_data.Id WHERE cars_data.Weight < 'terminal' EXCEPT SELECT DISTINCT model_list.Model FROM car_makers JOIN model_list ON car_makers.Id = model_list.Maker WHERE car_makers.Maker = 'terminal'
Prediction (Spider+Prompting OpenAI - text-davinci-003:SELECT Model FROM model_list JOIN cars_data ON model_list.ModelId = cars_data.Id WHERE Weight < 3500 AND Maker != 'Ford Motor Company' - (Missing other joint constraints)

Gold: SELECT DISTINCT T1.model FROM MODEL_LIST AS T1 JOIN CAR_NAMES AS T2 ON T1.Model = T2.Model JOIN CARS_DATA AS T3 ON T2.MakeId = T3.Id JOIN CAR_MAKERS AS T4 ON T1.Maker = T4.Id WHERE T3.weight < 3500 AND T4.FullName != 'Ford Motor Company'
Error: Due to the complex structure of the question in terms of the number of table involved, predicted query looks up at a wrong column for manufacturer name (in case of roberta). This could be due to not enough details provided in the question which makes even the prompting method fail.

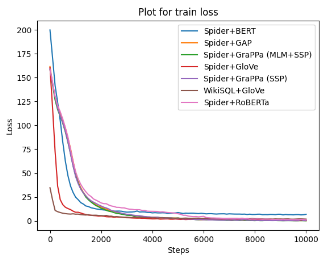
Table 13: In comparison with Figure 2 where the output for SPIDER+BERT (one of the baseline) and errors provided. We list down the experiments performed using RoBERTa and inference only experiment using few shot prompting

	Easy	**Medium**	**Hard**	**Extra Hard**	**All**
TRAIN					
count	24	28	17	17	86
Execution Accuracy	0.833	0.357	0.176	0.118	0.407
Exact Matching Accuracy	0.833	0.321	0.118	0.059	0.372
DEV					
count	3	3	3	1	10
Execution Accuracy	1.000	0.333	0.000	1.000	0.500
Exact Matching Accuracy	1.000	0.333	0.000	1.000	0.500

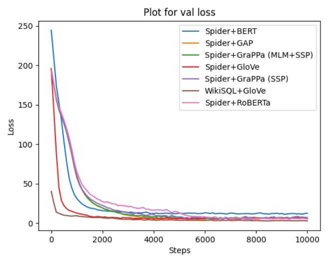
Table 14: GPT 3.5 Results on randomly sampled training and development examples



(a) Overall Loss Plot 0-10000 iterations

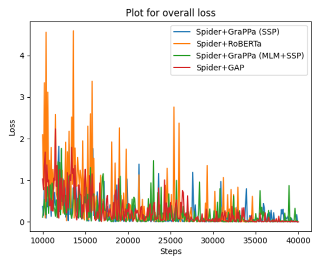


(b) Loss Plot for Train 0-10000 iterations

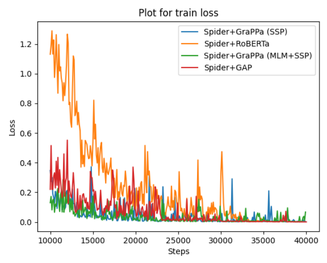


(c) Loss Plot for Val 0-10000 iterations

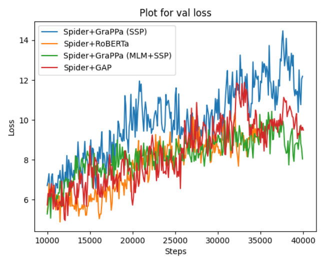
Figure 8: Loss Plots for the initial 10000 iterations



(a) Overall Loss Plot 10000-40000 iterations



(b) Loss Plot for Train 10000-40000 iterations



(c) Loss Plot for Val 10000-40000 iterations

Figure 9: Loss Plots for the 10000-40000 iterations

APPENDIX

A Additional Loss Plots

B Alignment

Few other alignment matrix snapshots:

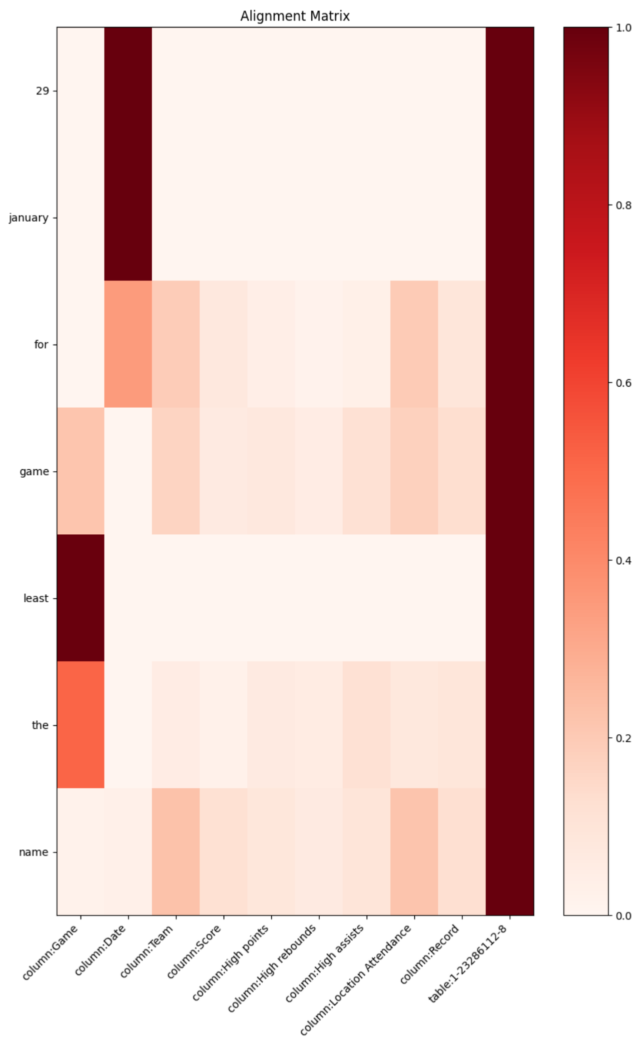


Figure 10: Alignment matrix for a question, inferred on WikiSQL+GloVe

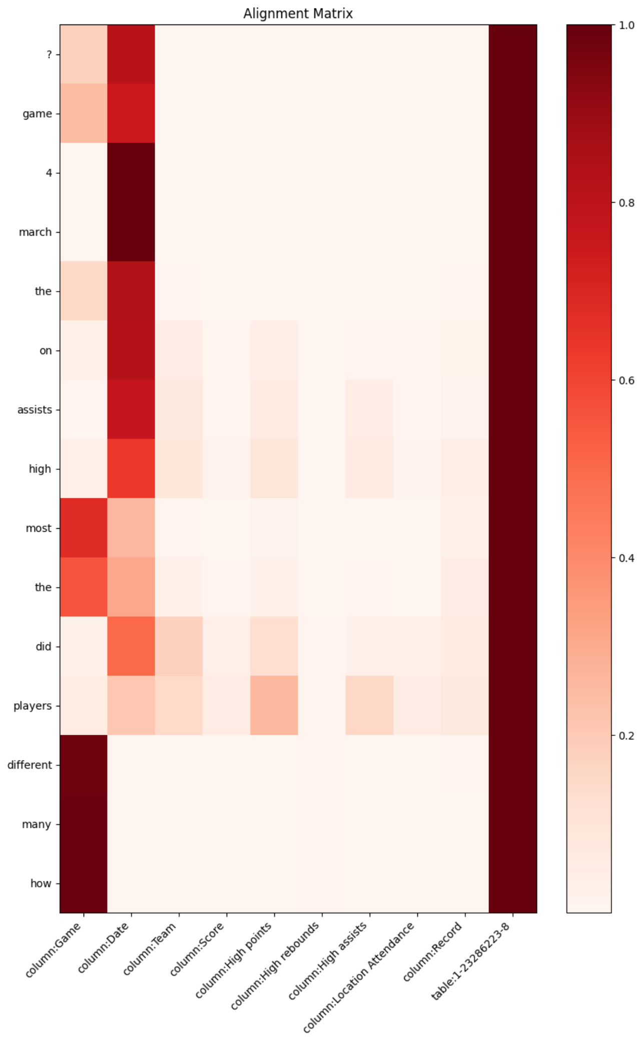


Figure 11: Alignment matrix for a different question, inferred on WikiSQL+GloVe

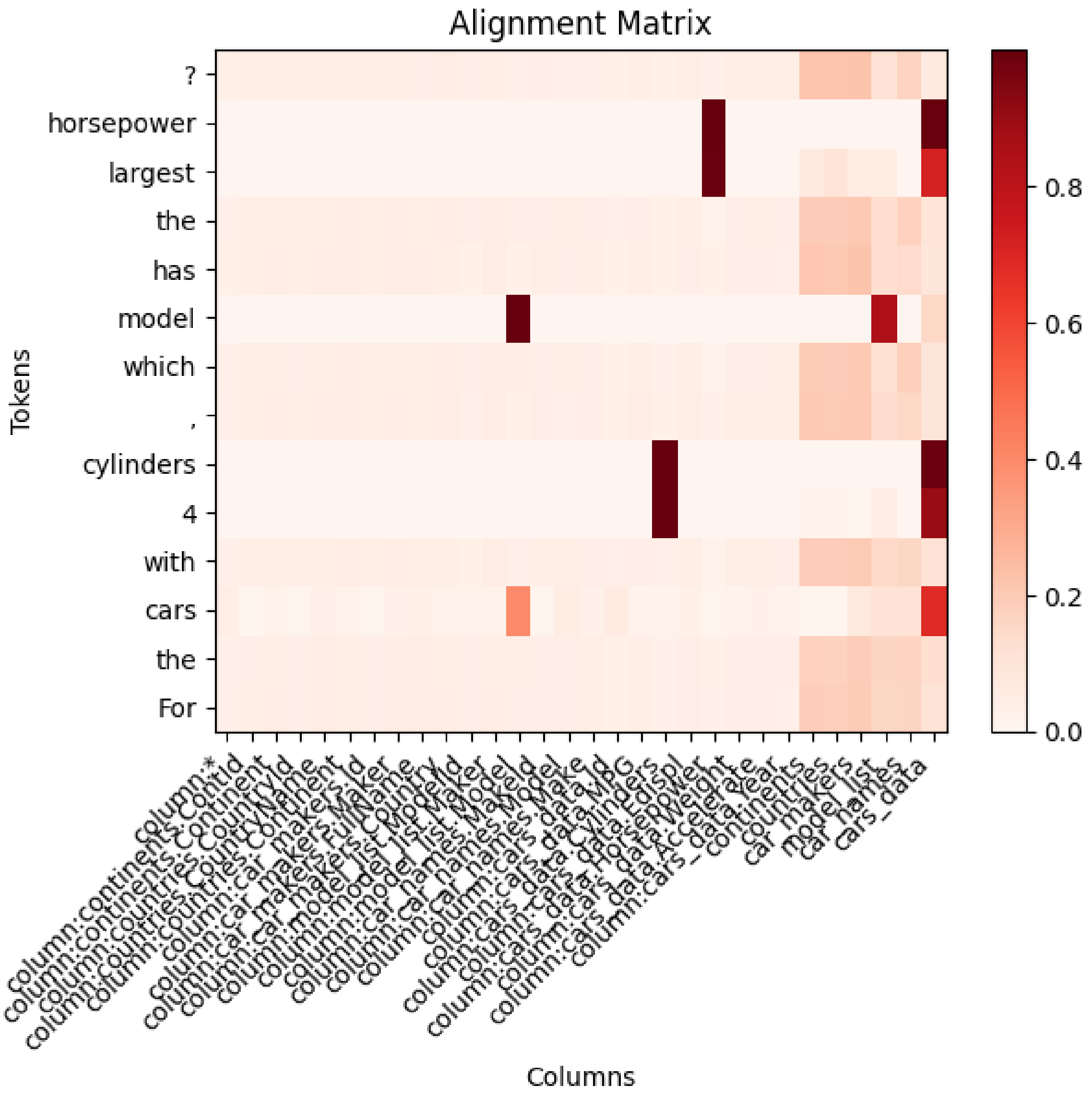


Figure 12: Alignment matrix for a question, inferred on Spider+BERT

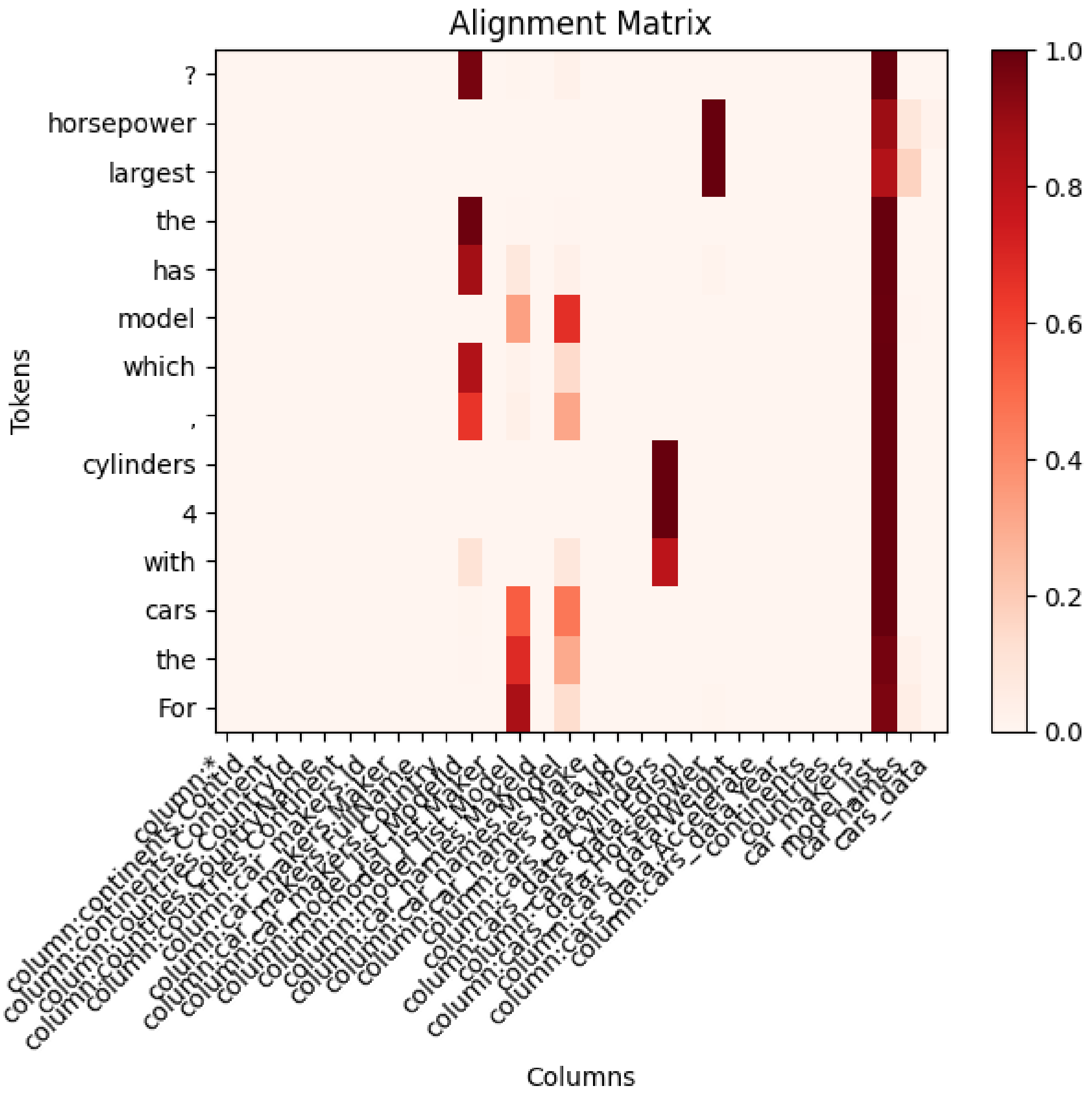


Figure 13: Alignment matrix for a question, inferred on Spider+GAP

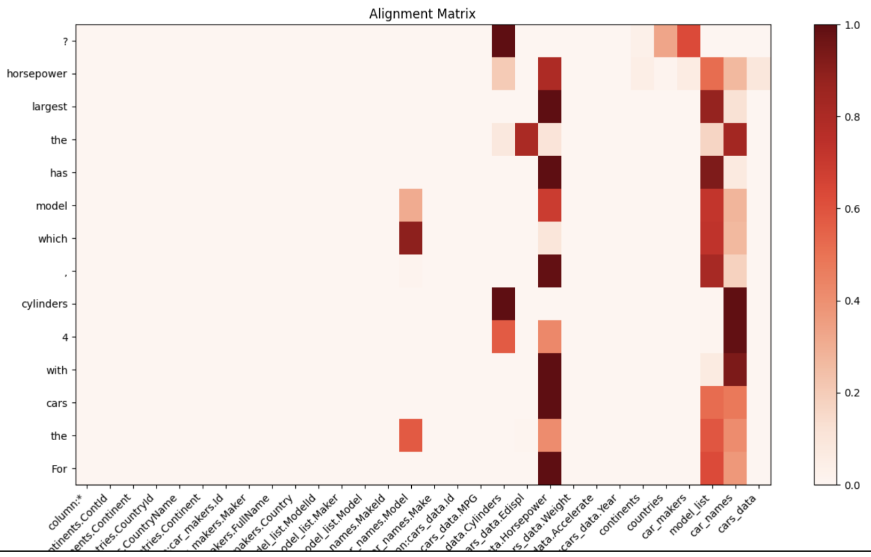


Figure 14: Alignment matrix for a question, inferred on Spider+GraPPa + SSP

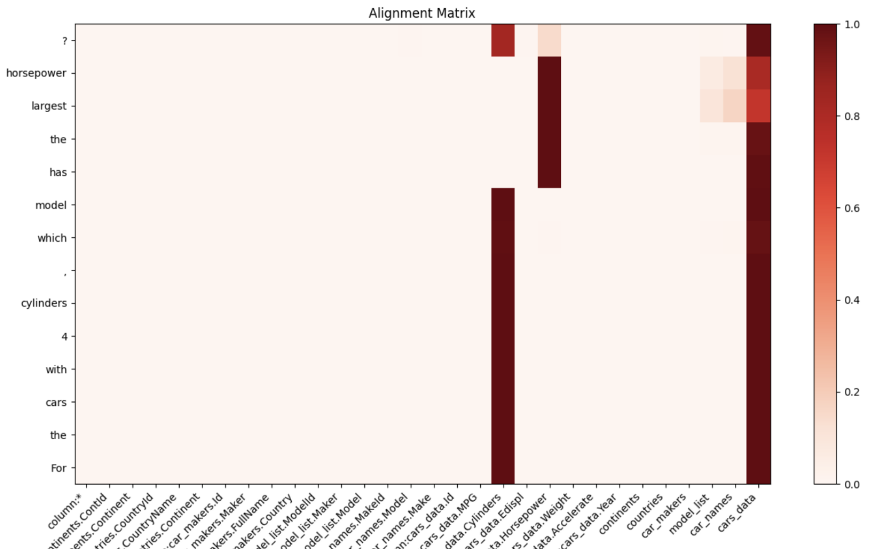


Figure 15: Alignment matrix for a question, inferred on Spider+GraPPa+MLM+SSP