Putting LLMs to Work: Text-To-SQL Generation

Roxy Rong
roxyrong@berkeley.edu

Jeremiah Stone
jeremiahstone@berkeley.edu

Abstract

The task of text-to-SQL translation has the potential to increase accessibility of data by allowing non-technical users to more easily interact with structured databases. There is currently significant research and study of various architectures to improve generalization and accuracy of generated queries. We explore the technique of soft prompt tuning to improve text-to-SQL translation performance.

1 Introduction

The growing emphasis on data analysis in the business sector is projected to increase the demand for skilled data analysts by over 23%[1] in the next decade, far outpacing the available supply by 50%[2]. Traditional education and training are unlikely to meet this demand, leading to the exploration of alternative solutions to increase data analysis accessibility such as enabling tools which lower the skill barrier for performing data analysis.

One such promising avenue to increase accessibility of enterprise data are text-to-SQL models to create a natural language query interface for relational business databases. Natural language query capabilities would allow workers to ask a database a question in the same way they might ask a colleague, opening up the possibility of self-service data access and analysis from ad-hoc queries to self-service data visualization. (Kumar et al., 2021)

Despite the promise of text-to-sql as a general purpose technology, it remains an immature and a very active area of research and development. Approaches to utilizing deep-learning techniques to perform sequence-to-sequence translation of text to SQL vary widely and there is currently no widely accepted best practice. The popular Yale Semantic Parsing and Text-to-SQL Challenge Leaderboard[3] lists dozens of techniques and architectures of varying complexity.

Our study explores the use of soft-prompt tuning compared to a full parameter fine-tuning with hard prompts for text-to-SQL translation. We analyze the effect of soft prompt tuning on models with both encoder-decoder (i.e. T5) and decoder-only structure (i.e. GPT-J). To better understand the implementation details, we experimented on variations of token input size and soft prompt initialization methods and compared their performance.

We hope our study can assist researchers and practitioners to decide between a full parameter fine-tuning and soft prompt tuning for their individual tasks. Compared to most text generation tasks, code generation tasks such as text-to-SQL have much stricter syntax rules, while requiring understanding the semantics of the input. Gaining insights into the influence of soft-prompt tuning on text-to-SQL dataset will enhance our understanding of what soft prompt tuning is capable of in similar code generation tasks.

Despite the potential of soft prompt tuning in code generation tasks, we concluded that soft-prompt tuning was inferior to fine-tuning specifically for text-to-SQL translation. With fine tuning we were able to achieve 2019 SOTA performance[4].

2 Background

Text-to-SQL can be broadly considered a variant of automated program generation, a field with deep history and a significant body of research. The current SOTA for code generation in general and SQL generation specifically is based on deep learning language-model based methods which overcome structural challenges with prior rule-based and complex parsing approaches. (Arcadinho et al., 2022) (Nan et al., 2023).

[1] https://www.bls.gov/ooh/math/operations-research-analysts.htm

[2] https://engineeringonline.ucr.edu/blog/data-scientist-shortage

[3] https://yale-lily.github.io/spider

[4] See footnote 3

Prior to the introduction of the Spider dataset (Yu et al., 2019), text-to-SQL conversion appeared to be a well solved problem with published results reaching over 80% translation accuracy on benchmarks such as ATIS and GeoQuery. The introduction of the Spider dataset and domain-specific quality scores renewed interest in the field, shifting from narrow training and testing on single domain datasets and databases to a complex multi-domain, multi-database evaluation model designed to test semantic parsing and overall generalization across domains and databases.

The current state of the field is broad, diverse, and active. Architectures range from intricate blending of transformer architecture coupled with sophisticated graph architectures to perform knowledge grounding such as Graphix-T5 (Li et al., 2023) to complex prompt engineering strategies utilizing pretrained and closed commercially available LLMs (Nan et al., 2023). Older techniques such as schema linking (Lei et al., 2020) or SQL output validation via constrained language model decoding such as PICARD (Scholak et al., 2021) are also widely adopted.

In our literature review, we found that one key area of debate is the ability for LLMs to perform given the lack of grounding, and many of the top performing approaches utilize structured knowledge grounding architectures. In parallel, however, the broader LLM community has continued to increase the quality of LLM output through focus on prompt engineering. The recent introduction of soft prompt tuning in particular seems promising to provide additional pre-trained context to LLMs to improve performance, especially when the base models are large (Lester et al., 2021).

In this work we seek to explore the potential of soft prompt tuning utilizing T5 (Raffel et al., 2020) finetuned to the task of text-to-SQL, compared with promising approaches of Soft Prompt Transfer (Vu et al., 2022) and decoder-only architectures with GPT-J (Wang and AI, 2021) and GPT-3.5-Turbo[5].

3 Methods

Our approach follows the experimental design which we have identified in current literature. Using the Spider data composed of natural language questions, data schemas, and target queries, we run a series of tests focused on translation accuracy between a natural language question and an SQL query. We perform a series of tests using T5-base as our target base model and also compare results to baseline GPT-J and GPT-3.5-Turbo models to gauge efficacy of our approach versus an alternate architecture. Our intuition is that a stepwise approach to varying model training, fine-tuning, and prompt strategies will yield insight into effectiveness of soft-prompt tuning in the domain of text-to-SQL translation. Finally, we are able to compare results collated on the Spider 1.0 leaderboard which tracks performance since 2018.

3.1 Dataset

Our experiments were mainly focused on Spider (Yu et al., 2019), a large multi-domain and cross-database dataset for text-to-SQL parsing. In our study, we used 7,000 samples for training, validated our model with 1,659 samples, and, using libraries provided by the Spider team, tested the model on 1,034 samples. Spider also provided a hidden test set, however, it requires us to submit our model in order to get the evaluation result. So we chose to evaluate the development set for ease. There is no overlap between questions or databases between the respective training, development, and test sets.

3.2 Evaluation

On Spider, we determine model performance based on two metrics: exact-set-match accuracy (EM), execution accuracy (EX). Exact-set-match accuracy compares the predicted and the ground-truth SQL query by parsing both into a normalized data structure. This comparison is not sensitive to literal query values and can decrease under semantic-preserving SQL query rewriting. (Scholak et al., 2021) Execution accuracy compares the results of executing the predicted and ground-truth SQL queries on the database contents shipped with the Spider dataset. This metric is sensitive to literal query values, but suffers from a high false positive rate (Zhong et al., 2020).

3.3 Experiments

3.3.1 T5 Baseline

Given that T5 was not designed for SQL generation, its performance on text-to-SQL questions in the Spider dataset would likely yield accuracy near zero. As a result, we decided to establish a fine-tuned model from T5 as our benchmark. We chose to use the T5-base model in an effort to control size

[5] https://platform.openai.com/docs/models/gpt-3-5

and cost of the model in line with our experimental objectives.

Our T5 prompt consists of a prefix ("Translate English to SQL"), a query, and the database schema associated with the query. Therefore, it is a zero-shot setting. We use the same basic prompt construction for all experiments in our study. Our T5 baseline was fine-tuned through training on the Spider dataset for 5 epochs, utilizing a batch size of 16 and a learning rate of 0.001 for each epoch. The execution accuracy and exact match accuracy is 0.423 and 0.427 respectively.

3.3.2 Fine-tuning T5

We implemented full fine-tuning with the same parameters of our T5 baseline, except that we trained over 10 epochs. We achieved execution accuracy and exact match accuracy of 0.456 for each. In subsequent experimentation with soft-prompt tuning, our goal was to contrast our results with this fully fine-tuned model, to assess whether soft-prompt tuning could achieve the results of full fine-tuning.

In the fine-tuning phase, we experimented with beam search during inference, setting the number of beams to 1, 2, and 4. Although increasing the beams to 2 or 4 slightly improved accuracy, the increment wasn't substantial enough to justify doubling our inference time. Additional strategies, such as validating SQL output for each beam, could potentially improve the results, but this is beyond the scope of our research. Consequently, we opted for a single beam during inference in all our experiments.

3.3.3 Soft-Prompt Tuning T5

We applied the soft-prompt tuning technique to our T5 base model and initiated the weights of 100 tokens randomly. This represents 76,800 parameters, 0.0344% of the total parameters of the T5 model. The training lasted for 25 epochs, utilizing a batch size of 16 and a learning rate of 0.3. We also included the AdaFactor optimizer with weight decay 1e5, β_2 decay 0.8, and parameter scaling off (Shazeer and Stern, 2018). Soft prompt tuning is known for slow convergence. After 25 epochs, even though validation loss kept dropping and our early stopping callbacks were not hit, we stopped further training due to the lack of improvement in accuracy.

3.3.4 Initialization Method for Soft-Prompt Tuning

In an effort to delve into the effects of initialization on soft-prompt tuning, we conducted experiments with several initialization techniques, as described below. We used each set of initialized weights for training over 5 epochs, while keeping other hyperparameters constant.

- Random Initialization: This method sets the soft prompts to random values at the beginning of the training process.
- Tokenizer Vocab Sampling: This method involves selecting words randomly from the tokenizer's vocabulary and concatenating them.
- Top 500 Question Tokens: This method uses the most frequent 500 words from the prompt portion of the dataset, excluding stopwords and punctuation.
- Top 500 Query Tokens: This method uses the 500 most frequent words from the SQL output of the dataset, excluding stopwords and punctuation.
- "Prefix" Initialization: This initialization is aligned with the hard prompt used for T5 fine-tuning, specific to our use case, which is "Translate English to SQL."

3.3.5 Token Size for Soft Prompt Tuning

We further evaluated the performance of soft prompt tuning by examining different token sizes. Specifically, we experimented with token sizes of 20, 50, and 100, and conducted training for 5 epochs, while keeping all other hyperparameters constant.

3.3.6 Decoder-only Models

After examining the encoder-decoder model such as T5, we picked GPT-J as our starting point for decoder-only models. Similar to T5, GPT-J was not originally built for SQL creation. This model, containing 6B parameters and requiring 48G RAM for inference, led us to opt for the 8-bit variant of GPT-J. Starting from a zero-shot setting, we found that the generated tokens were a blend of text with SQL and continued to generate irrelevant content, if stopping criteria was not imposed.

We made efforts to enhance the prediction by exploring one-shot and few-shot setups with diverse prompt structure and incorporating stopping

rules during inference. Even though these changes increased the chance of valid SQL output, the accuracy was still quite low (.05 execution accuracy, .07 exact match).

3.3.7 GPT models

After exploring GPT-J, we turned to GPT models from OpenAI. These models are only available via API. GPT-4 is not available until a successful payment of $1 is made to OpenAI, which we were not able to achieve during our project timeline.

We constructed an instruction-oriented prompt for GPT-3.5-Turbo, utilizing the same test set and tabular representation of the schema as in other experiments.

4 Results

For reporting, we use the average of execution accuracy (EX) and exact-matching accuracy (EM) as our metrics.

4.1 Baseline, Fine-tuning and Soft-Prompt Tuning

Our task accuracy is shown in Figure 1. The blue line illustrates an accuracy of 0.425 achieved after fine-tuning T5 for 5 epochs, where we established our baseline. The red line represents our result of further fine-tuning, attaining an accuracy of 0.456 after an additional 5 epochs of training. Soft prompt tuning, denoted by the yellow line, did not contribute substantially to the model's improvement, resulting in an accuracy of only 0.431 after training for a total of 10 epochs.

4.2 Prompt Initialization

As shown in Table 1, We examined the differences in execution and exact matching accuracy across various initialization techniques. Among all the approaches, methods incorporating random elements like Random Init and Tokenizer Vocab Sampling seemed to yield better outcomes. Instead of selecting the most frequent tokens from the input or labels, the random initialization appears less prone to over-fitting. The approach using the Top 500 Question Token yielded notably inferior results. One possible reason for this could be that the repetitive schema in the prompt is not conducive to effective initialization.

Our findings diverge from previous research, such as that by Lester et al., 2021, which indicated that random initialization is less effective compared to more "advanced" initialization methods, particularly in tasks centered around Question and Answering. Our differing conclusions may stem from the unique nature of our task and the use of a comparatively small base model for fine-tuning. Additionally, it is important to note that owing to the absence of improvement in soft-prompt tuning for the text-to-SQL task, our conclusion should not be generalized to other tasks.

Init Method	EX	EM
Base model	0.423	0.427
Random Init (1)	0.423	0.429
Tokenizer Vocab Sampling	0.417	0.428
Top-500 Question Tokens	0.349	0.351
Top-500 Label Token	0.410	0.426
Prefix	0.327	0.328

Table 1: accuracy over initialization methods

4.3 Prompt Token Size

Our findings into the best number of tokens for soft-prompt tuning is limited, as shown in table 2, though in line with previous research. We chose to utilize 100 tokens for soft-prompt tuning since this number yielded the best results after training for 5 epochs in our experiment. From prior studies (Lester et al., 2021), we also learned that employing more than 100 tokens in a soft prompt ceases to enhance performance.

num_of_tokens	EX	EM
20	0.420	0.426
50	0.417	0.426
100	0.423	0.429

Table 2: Accuracy metrics for different numbers of tokens

4.4 Beam Search

We opened the beam up and performed it during our fine-tuned model inference. The results are shown in Table 3. Without components such as PICARD or T5QL to better rank the number of outputs and eliminate the un-executable ones , the enhancement was slight, and there was no further improvement observed after using 4 beams.

4.5 GPT-J Experiments

Because of the low accuracy in both the GPT-J 8-bit base model and the fine-tuned model using

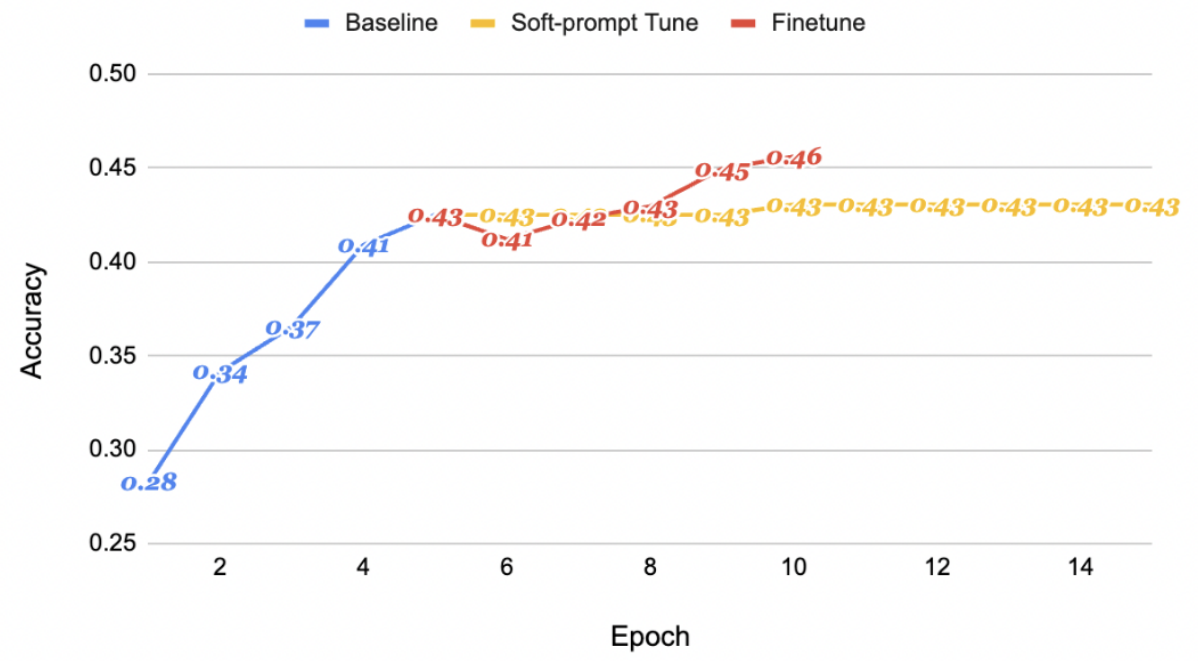

Figure 1: Text-to-SQL task accuracy for baseline, soft-prompt tuning and finetuning

num_of_beams	EX	EM
1	0.456	0.456
2	0.463	0.464
4	0.462	0.462

Table 3: Accuracy metrics for different numbers of beams

zero, one-shot, and two-shot settings, we are not reporting the specific results here. Nevertheless, we have made progress: from nearly no valid SQL output in the zero-shot setting in the base model, we managed to create 82% valid SQL output using the two-shot setting, coupled with a working stopping criteria.

4.6 GPT 3.5 Turbo Experiments

We utilized the OpenAI API for GPT-3.5-Turbo. In order to maximize repeatability and minimize extraneous results, we set temperature=0 and did not change any other default parameters to the model. As expected from reading the literature, GPT-3.5-Turbo performed well, achieving Exact Match and Execution Accuracy of .416 and .564 respectively.

4.7 Error Analysis

We examined the mistakes made by the top-performing fine-tuned T5 model and GPT-3.5-Turbo by identifying patterns in the intersection of failed prompts. Our findings reveal that the common errors typically involve complex structural dependencies, necessitating a more sophisticated understanding of tables. In the example provided below, the term "highest average" is unclear, as there is an average column in the stadium table. This ambiguity was not recognized by either Finetuned-T5 or GPT-3.5-Turbo.

- : Question: What is the name and capacity for the stadium with highest average attendance?
- : Schema:

```
| stadium : stadium_id , location ,
name , capacity , highest , lowest
, average | singer : singer_id
, name , country , song_name ,
song_release_year , age , is_male |
concert : concert_id , concert_name
, theme , stadium_id , year |
singer_in_concert : concert_id ,
singer_id
```

- Gold:

```
SELECT name , capacity
    FROM stadium ORDER BY
    average DESC LIMIT 1
```

- T5 pred:

```
SELECT name, capacity FROM
    stadium GROUP BY
   stadium_id ORDER BY
   avg(attendance) DESC
   LIMIT 1
```

- : GPT3.5 pred:

```
SELECT name, capacity FROM
    stadium WHERE average
    = (SELECT MAX(average
   ) FROM stadium)
```

4.8 Experiment Summary

Our results indicate that while fine-tuning delivered significant performance improvement relative to baseline, soft prompt tuning did not. Interestingly, however, despite being a significantly larger SOTA language model which, we can infer from OpenAI documentation, is trained on software code which in all probability includes copious SQL, GPT-3.5-Turbo also did not manage better than 56% execution accuracy. This indicates that there is an upper bound to how well a non-grounded model can perform in text-to-SQL translation. The success of efforts to introduce grounding such as those in graphix-t5 indicate that this pathway holds the most promise for a best practice in the task of text-to-SQL generation.

5 Conclusion

In this study we explored the use of soft prompt tuning to enhance the performance of a tuned T5-base model to perform text-to-SQL translation, comparing our results with T5 fine tuning and decoder models GPT-J and GPT-3.5 turbo. We found that soft prompt tuning did not result in a significant performance against the task and that fine tuning and GPT-3.5-Turbo out performed the soft prompt tuning approach.

References

Samuel Arcadinho et al. 2022. T5ql: Taming language models for sql generation. In *Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 276–286.

Ayush Kumar et al. 2021. Deep learning driven natural languages text to sql query conversion: A survey. *JOURNAL OF LATEX CLASS FILES*, 14(No. 8).

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-sql. In *EMNLP 2020*.

Brian Lester et al. 2021. The power of scale for parameter-efficient prompt tuning. In *EMNLP 2021*.

Jinyang Li et al. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. In *AAAI 2023 main conference (oral)*.

Linyong Nan et al. 2023. Enhancing few-shot text-to-sql capabilities of large language models: A study on prompt design strategies.

Colin Raffel et al. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. In *Accepted to EMNLP 2021*.

Noam Shazeer and Mitchell Stern. 2018. Adafactor: Adaptive learning rates with sublinear memory cost. In *Proceedings of the 35th International Conference on Machine Learning*.

Tu Vu et al. 2022. Spot: Better frozen model adaptation through soft prompt transfer. In *Accepted as a main conference paper at ACL 2022*.

Ben Wang and Eleuther AI. 2021. Gpt-j-6b: 6b jax-based transformer – aran komatsuzaki. Aran Komatsuzaki. Accessed 5 August 2023.

Tao Yu et al. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP 2018*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. In *EMNLP 2020*.

6 Appendix

Listing 1: T5 Prompt Construction

```
# construct T5 prompt
prefix = 'translate␣English␣to␣SQL:'
dev_spider['schema'] = dev_spider['db_id'].map(schema_dict)
dev_spider['prompt'] = prefix + dev_spider['question'] + '\nDatabse␣schema␣is␣'
                       + dev_spider['schema']
```

Listing 2: GPT-J Few Shot Prompt Construction

```
# Two-shot training style prompt
fixed_few_shot_prefix =
            """Context: | ref_document_types : document_type_code ,
            document_type_description | roles : role_code , role_description
            | addresses : address_id , address_details | ref_document_status
            : document_status_code , document_status_description
            | ref_shipping_agents : shipping_agent_code , shipping_agent_name ,
            shipping_agent_description | documents : document_id ,
            document_status_code , document_type_code , shipping_agent_code ,
            receipt_date , receipt_number , other_details
            | employees : employee_id , role_code , employee_name , other_details
            | document_drafts : document_id , draft_number , draft_details
            | draft_copies : document_id , draft_number , copy_number
            | circulation_history : document_id , draft_number , copy_number ,
            employee_id | documents_mailed : document_id , mailed_to_address_id ,
            mailing_date
           Question: Which employee has showed up in most circulation history
           documents?
           List the employee's name and the number of drafts and copies.\n"
           Answer: SELECT Employees.employee_name , count(*) FROM Employees
           JOIN Circulation_History ON Circulation_History.employee_id =
           Employees.employee_id
           GROUP BY Circulation_History.document_id ,
           Circulation_History.draft_number ,
           Circulation_History.copy_number
           ORDER BY count(*) DESC LIMIT 1;
           ###
           Context:| member : member_id , card_number , name , hometown , level
           | branch : branch_id , name , open_year , address_road , city ,
           membership_amount
           | membership_register_branch : member_id , branch_id , register_year
           | purchase : member_id , branch_id , year , total_pounds
           Question: What are names for top three branches with most number of
           membership?
           Answer: SELECT name FROM branch ORDER BY membership_amount DESC LIMIT 3
           ###
           Context: """

fixed_few_shot_infix = "\n␣Question:␣"
fixed_few_shot_postfix = "\n␣Answer:␣"

dev_spider['schema'] = dev_spider['db_id'].map(schema_dict)
dev_spider['prompt'] = fixed_few_shot_prefix + dev_spider['schema']
                       + fixed_few_shot_infix + dev_spider['question']
                       + fixed_few_shot_postfix
```

Listing 3: GPT-3.5-Turbo Instruction-Based Prompt Construction

```
def create_prompts(row):

  instruction_prefix = {'role': 'system', 'content':"""
  Parse the question provided into SQL based on a provided schema
  which describes a database schema.
  Based on the provided schema, create an ANSI-92 SQL Query to answer the provided
  Return the answer as a single line SQL query ONLY. Do not include any additional
  explanation."
```

```
    """}

    instruction_infix = {'role': 'system', 'content':"""
    Schema: """ + row['schema']}

    instruction_postfix = {'role': 'user', 'content':"""
    Question: """ + row['question']}

    return[instruction_prefix, instruction_infix, instruction_postfix]

# Update dev_spider with prompts
dev_spider['schema'] = dev_spider['db_id'].map(schema_dict)
dev_spider['context'] = dev_spider.apply(create_prompts, axis=1)
print(dev_spider['context'][500])
```