CodeT5 Fine-tuned on Text-To-SQL

Wasiim Ouro-Sama and Jacob Austin

Abstract

Pretrained Seq2Seq Natural Language Processing models such as T5, have been recently shown to extend well to programming languages as evidenced by the performance of the CodeT5 model trained on NL-PL pairs. In this paper, we attempt to extend the base CodeT5 model to NL-SQL translation trained on the spider dataset. Our experiments show that our model approaches the performance of certain models which leverage the SQL syntax structure as opposed to solely using data found in the dataset.

1 Introduction & Related Works

Building a natural language model that is able to map natural language queries to corresponding SQL code is a challenging task. Not so much in terms of generating appropriate syntax, but more so in being able to map certain words and values to corresponding tokens that are valid and exist within the database schema and to be able to generalize outside of schemas used in training. Therefore special care must be taken at the preprocessing and postprocessing stages to ensure the model produces as valid an input as possible based on the schema. The dataset being used is also of particular importance, if we want the model to learn complex foreign key relationships the dataset must include such.

Prior attempts at NL-SQL translation have been made in the past. A comparable result in terms of performance to our CodeT5 model is TypeSQL that takes a slot-filling approach. TypeSQL is pre-trained on the WikiSQL dataset. Taking into consideration the fact that natural language questions often contain entities and numbers that are specific to the underlying database. Previous works showed that such words are crucial to many downstream tasks, but pre-trained models are usually poor at producing accurate embeddings for these words. The solution proposed is to assign each word a type, for example the name “mort ducker” would get assigned to PERSON, this type information being extracted from some knowledge base. “Title”, “issue” assigned COLUMN from the schema.

As for methodology, TypeSQL, much like its predecessors, formats the task as a slot-filling problem. In the image below the objective would be to replace the tokens starting with a ‘$’ sign with the appropriate token for that type as can be seen in Figure 1.

```
SELECT $AGG $SELECT_COL
WHERE $COND_COL $OP $COND_VAL
(AND $COND_COL $OP  $COND_VAL)*
```

Figure 1. TypeSQL slot filling approach

The questions are first preprocessed and labeled by their appropriate type as mentioned above. Then a two bi-directional LSTM is used to encode words in the question with their types and associated column names separately. The motivation for an LSTM is the fact that it is able to capture order dependence in a sequence prediction problem like SQL generation. The output hidden states of this are then used to predict values for the slots. TypeSQL achieves great performance on the WikiSQL dataset, whereas it does not do so well on the the spider dataset, which would suggest it had a difficult time dealing with the more complex queries in the spider dataset as compared to the very simple and identical queries in the WikiSQL dataset.

Our dataset of choice is the Spider dataset. The particular improvement this dataset offers is that it consists of a variety of databases with multiple tables, a wide variety of handwritten SQL queries which include many complex SQL clauses such as joins and nested queries forcing the model to learn relationships between tables and columns in the database schema.

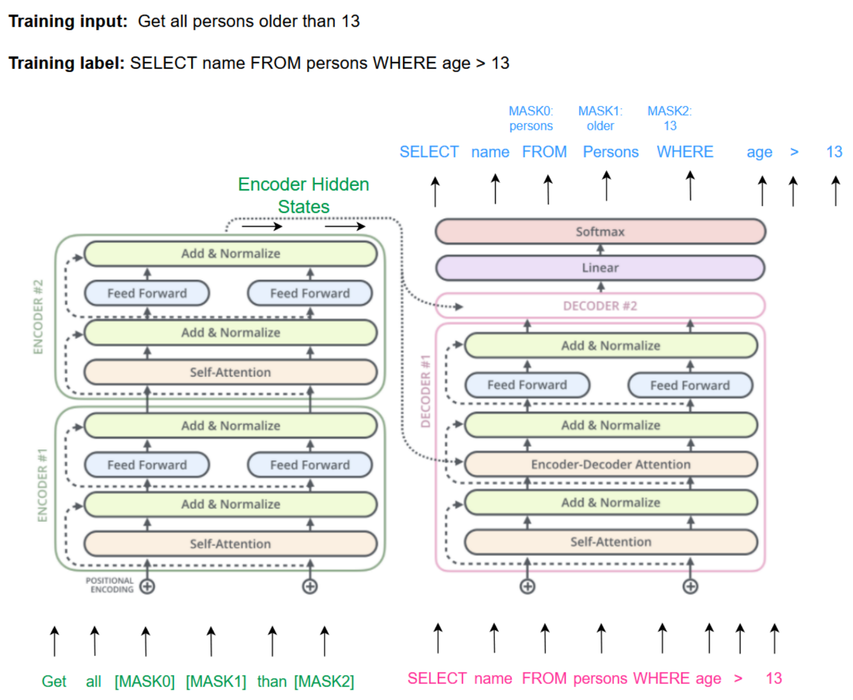

Training input: Get all persons older than 13
Training label: SELECT name FROM persons WHERE age > 13
MASK0: persons
MASK1: older
MASK2: 13
SELECT name FROM Persons WHERE age > 13
Encoder Hidden States
Softmax
Linear
DECODER #2
DECODER #1
ENCODER #2
ENCODER #1
Add & Normalize
Feed Forward
Self-Attention
Encoder-Decoder Attention
POSITIONAL ENCODING
Get all [MASK0] [MASK1] than [MASK2]
SELECT name FROM persons WHERE age > 13

Figure 2. CodeT5 model finetuned for Text-To-SQL

Another paper is PICARD, this is the most comparable work in terms of methodology to what we are doing. PICARD attempts to finetune the T5 model on the NL-SQL task. This paper makes a modification to the standard beam search procedure called the Picard method. Picard has access to database schema information, in particular, information about the names of tables and columns within them. Hence, Picard is able to account for invalid column names or nonexistent columns during the generation process. It is also able to convert the model outputs into an AST form to check for the validity of the SQL query syntax. Surprisingly, they found that a T5-base model with Picard can outperform a T5-large model without it, and likewise for a T5-large, and a T5-3B model and raised to the state-of-the-art performance on the Spider dataset.

2 CodeT5 Model

Our work draws on prior exploration of adapting T5 to NL-PL, and PL-PL translation, that is the CodeT5 paper. The primary contribution of this work are the tasks on which the CodeT5 model is pre-trained on, and the deliberate masking of identifiers in the Masked Identifier Prediction Task, identifier tagging and bimodal NL-PL, and PL-NL language generation that makes this model particularly fit for NL-PL translation over T5. We simply pass our input natural language to this and compute loss over the set of logits produced by the model. A diagram of the model can be seen in Figure 2.

2.1 Dataset

We use the Spider Dataset which consists of 7000 NL-SQL pairs in the training set and 1034 rows for evaluation. In addition to the previously mentioned improvements made by the Spider dataset. The dataset is also split in such a way such that there is no database overlap in the train and test.

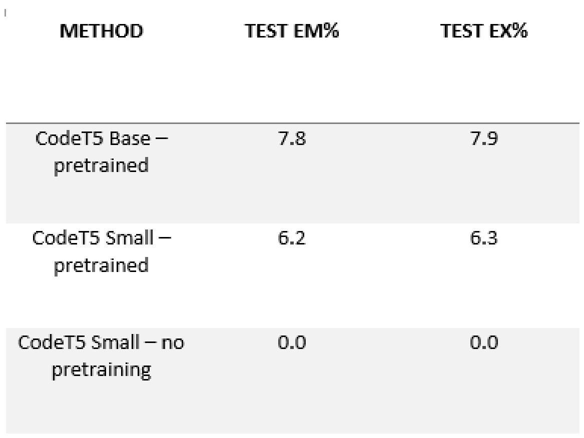
2.2 Experiment

We train the CodeT5 base model on the spider development set and evaluate on the test set.

On Spider, we evaluate model performance according to four metrics. Exact-set-match accuracy, execution accuracy (with values), execution accuracy (without values), as well as a sacreBLEU score. Exact set-match compares predicted and ground truth token for token and does not account for out of order values leading to false-negatives, but this can act as a lower bound. Execution accuracy compares the results of executing the predicted and ground truth on databases that come with the spider dataset, idea being if both produce the same results across the databases in question, they are equivalent, this aims to provide the best possible approximation of semantic accuracy. There are two variations, with values, that substitutes values from the ground truth into the predicted, and without values and therefore less concerned about literal values than execution without values. However we find that both variations yield almost similar results, which would suggest that the model is able to predict the correct values in most cases. In general we find the execution match to be more accurate, although it is prone to false positives, since we are checking against an entire suite of databases for each query, this is minimal.

3 Results

Our results for the base and small models of CodeT5 are compared in Figure 3, compared to a baseline model without pretraining.



METHOD	TEST EM%	TEST EX%
CodeT5 Base – pretrained	7.8	7.9
CodeT5 Small – pretrained	6.2	6.3
CodeT5 Small – no pretraining	0.0	0.0

Figure 3. Test EM% and EX% across our models

In this figure we can see that a small model with no pre-training was not able to achieve any exact matches or executions under the test conditions. For these comparisons in Figure 3 the pretrained models were run for 30 epochs and the unpretrained model was run for 5 epochs due to time constraints. However, we were able to achieve an EM% of 5.8% within 5 epochs on the small model and 6.8% within 5 epochs on the base.

For our hyperparameters, the base models were trained with a batch size of 16 and the small with a batch size of 32. Final numbers were run on models over 30 epochs, and learning rate was kept at 1e-4. Also for all models, we used a weight decay of 0.01, and 200 warmup steps.

As for the sacreBLEU scores, we were able to achieve 33.5 BLEU on the small model, and 35.9 BLEU on the base model. Due to time constraints during development, we were unable to obtain a BLEU score for the model without pretraining, although our focus is mainly on EM% for this paper.

Figure 4 shows a comparison to other models on the Spider Dataset Leaderboards. We compare to two of the models at the bottom of the leaderboards (Seq2Seq, TypeSQL), and the highest non-anonymous model without use of DB content (RYANSQL v2 + BERT).

METHOD	TEST EM%
CodeT5 Base (ours)	7.8
Seq2Seq + attention (Dong and Lapata, ACL '16)	4.8
TypeSQL (Yu et al., '18)	8.2
RYANSQL v2 + BERT (Choi et al., '20)	60.6

Figure 4. Test EM% across various models

The purpose of this comparison is to show current state-of-the-art percentage on the Spider dataset, along with a few models with comparable performance to our own. We can see that our model cannot compare to state-of-the-art solutions, but this is as expected considering our time and computing resource limitations. However, our model was comparable to some of the models listed on the dataset leaderboards, showing we did achieve some meaningful performance on the Spider dataset.

We show a comparison of our performance on different query difficulties in Figure 5. These difficulties are defined by Spider in the dataset.

METHOD	EASY EM%	MEDIUM EM%	HARD EM%
CodeT5 Base	18.15	6.05	4.60
CodeT5 Small	17.34	3.59	1.72

Figure 5. Test EM% across difficulties

In this figure we can see that EM% declines as difficulty of query rises, which is as expected. We can also see that the small model performs similarly at easier difficulties, with the base model performing only 4.7% better on easy queries. However there is a large separation on harder queries, with the base model performing 167.4% better than the small model on hard queries. With more time to tune parameters for the base model and more computing resources, we believe we could achieve much better performance with the base model overall.

4 Conclusion

Overall, while we were not able to achieve state-of-the-art production with our available setup, we had results that were comparable to some of the models on the Spider Dataset Leaderboards, and massively outperformed an unpretrained model under the same test conditions. Pretraining on code makes a large difference in our evaluation metrics, as features such as context and grammar are important for execution of SQL queries and take time to learn from a randomly instantiated model.

In our testing, we also show that the base model is able to outperform the small model on all levels of metrics, and particularly outperformed the small model on harder SQL queries. In the future, it would be ideal to test with CodeT5 Large, CodeT5 3B (3 billion parameters), and CodeT5 11B (11 billion parameters), although we are currently unable to test on these models due to computing resource limitations.

For future models we would also prefer to do additional fine-tuning of hyperparameters for CodeT5 Base. Most of our parameter testing was done on CodeT5 Small due to the two week time limit to complete code, so our hyperparameters for the base model could likely be improved with more testing.

In conclusion, we achieved meaningful results with the time and resources available, but further testing would help us improve both on our current model and investigate benefits of larger models on the dataset.