CodeT5-SQL

Investigating the Limits of Mapping Natural Language to SQL in Transformer Architectures

Matthew Ball and **Vivek Reddy**
University of California Berkeley School of Information
University Avenue and Oxford St, Berkeley, CA 94720
Matthew.Ball@Berkeley.edu and VReddy704@Berkeley.edu

Abstract

Training large language models (LLMs) on source code using the transformer architecture has improved flexibility and progress toward code intelligence. Existing state-of-the-art models use an encoder-decoder network to handle multiple downstream tasks such as code summarization, code generation, clone detection, etc. This paper focuses on text-to-code generation, specifically mapping natural language to appropriate SQL queries. We aim to design a robust model that can accurately write SQL queries to answer a question given several databases. We first designed a cross-domain SQL dataset from three of the largest text-to-SQL datasets on Hugging Face to train our models. Next, for our baseline, we took the CodeT5 model on Hugging Face, trained it on our SQL dataset, and saved the model with the best validation score for testing. We used five techniques to improve our baseline: curriculum learning, cross-domain datasets, Decomposed Low-Rank Adaptation (DoRA), multi-task learning, and hyper-parameter tuning. These methods improved the model's performance by 3% in Rouge score evaluations over the baseline.

1 Introduction

Today, businesses rely on an abundance of data stored in their traditional databases to make critical decisions. SQL is a common way to query, retrieve, and make sense of this information. However, writing lengthy SQL queries can be time-consuming, and learning to write them can also be a barrier to entry. Automating the natural language translation process to SQL queries can enable more people to work with the data.

With the advent of transformer architectures and increasing model capacity in recent years, model performance has improved. State-of-the-art methods for this task typically use sequence-to-sequence model architectures. Building on this, we have decided to use an encoder-decoder model, which allows us to first do NLP on the natural language query by extracting key elements and converting them into a structured format. These key elements are then fed into the decoder and mapped into SQL, and an appropriate SQL query is generated at the end of the model. To improve our model performance, we performed five experiments that have shown to work in code generation and wanted to see if they transfer to SQL generation.

1.1 Contributions

Our contributions include adapting the CodeT5 architecture to generate SQL queries from a natural language using five techniques: curriculum learning, cross-domain datasets, Decomposed Low-Rank Adaptation (DoRA), multi-task learning, and hyperparameter tuning. We ran each experiment individually and in parallel to test how it affected the original model. More details on each technique are in the metrics and results section below.

2 Background

Many approaches have been studied to address the Text-to-SQL problem since at least the 1970s (Deng et al.,2022). The challenge lies in correctly extracting the semantic meaning of a natural language question and translating it into a valid and correct SQL query. Some approaches have involved providing an SQL template to the model. This template allows the model to focus only on predicting the content since the grammar is provided. SQLNet (Xu et al., 2017) learns to predict only the content by filling in template slots, but its performance is limited when predicting nested queries. Other methods involved putting constraints on the decoder to always output syntactically valid SQL queries. PICARD (Scholak et al., 2021) is a method that discards outputs in the decoder beam if the parser determines it has reached an invalid point. More recently, CodeT5 (Wang et al., 2023), a variant of the T5 model, was introduced for code generation and summarization tasks.

2.1 Text-to-SQL

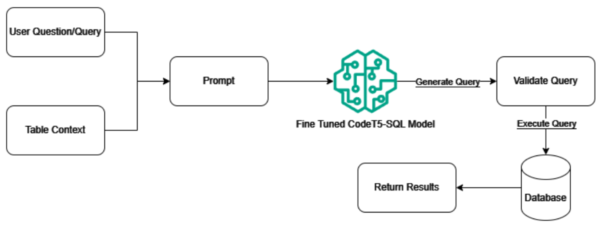


Figure 1: Process of Text-to-SQL

Text-to-SQL involves understanding and translating a natural language query into an SQL query. Most data is stored on traditional servers and accessed using SQL. To increase accessibility to this data, we approach this task by converting natural language questions into a structured representation that an LLM can use to generate a semantically correct SQL query that will be run on a database. However, no standard benchmark or dataset is currently used in the field, which prohibits the development of more effective and efficient machine learning models.

2.2 CodeT5

Prior methods for developing text-to-code machine learning models involved using an encoder/decoder-only model (BERT & GPT). The introduction of CodeT5 was one of the first implementations to use an encoder-decoder model to handle various code generation tasks (summarization, nl-to-pl, pl-to-nl, etc.). To accomplish this, CodeT5 used the T5 architecture and trained on a sequence of tasks to help capture semantic information specific to code: masked span prediction, identifier tagging, masked identifier prediction, and bimodal dual generation.

3 Datasets

For this project, we developed a merged dataset using three popular text-to-SQL datasets found on Hugging Face. This merged dataset was developed to prevent the model from overfitting.

3.1 Clinton/Text-to-SQL-v1 Dataset

This large cross-domain dataset comprises over 250,000 records from domains in healthcare, sports, government, and others. It has five columns: instruction, input, response, source, and text.

3.2 b-mc2/SQL-create-context Datase

A dataset of about 78,000 records was built on WikiSQL, a large dataset of simple one-table SQL queries, and Spider, a smaller dataset of more complex SQL queries. The dataset has three columns: question, context, and answer.

3.3 gretelai/synthetic_text_to_SQL

A synthetic text-to-SQL dataset was released in April 2024, with 105,000 records covering over 100 domains. This dataset has 11 fields: id, domain, domain_description, SQL_complexity, SQL_complexity_description, SQL_prompt, SQL_context, SQL, and SQL_explanation. Gretelai used the LLM-as-judge technique with GPT4 to test the data quality, comparing its SQL dataset to the b-mc2SQL-create-context dataset (mentioned above).

3.4 Merged Preprocessed Dataset

The merged dataset is created using the Clinton, b-mc2, and Gretelai datasets mentioned above. From each dataset, we take the question, database schema, and answer to form our dataset. The question and database schema are added to our pre-made prompt (Appendix 3) and outputted to a JSON file. The merged dataset is then separated into training, validation, and test datasets (80% training, 10% validation, and 10% test). This dataset is then cached and stored into a tensor file to load the data faster and run tests more efficiently. Pre-made prompts exceeding 512 tokens were dropped as they exceeded the capacity of T5's input sequence, and only a small percentage of examples exceeded this amount. The train validation split was also designed so that any database schema passed in during model training was not seen in the model validation

4 Baseline Model

We loaded the model and tokenizer for the CodeT5 220m parameter model off of HuggingFace. We trained this model to predict appropriate SQL queries using a prompt that included the question and the database context. For this model, we wanted to see how well the weights in CodeT5 would adjust to SQL, given that CodeT5 was trained on other programming languages. We were surprised to discover that even though we had over 300,000 examples in our self-made dataset, the

Model	Trainable Parameters	All Parameters	Trainable %
Code-T5 (w/o DoRA)	251,248,896	251,248,896	%100
Code-T5 (w/ DoRA)	28,366,848	251,248,896	%11.29

Table 1: Table of DoRA adjusted parameter sizes

base model always seemed to overfit (high validation loss and low training loss).

5 Experimental Methods

Our experiment explored five methods to enhance our model's ability to generate accurate SQL queries. These methods included curriculum learning, cross-domain datasets, Decomposed Low-Rank Adaptation (DoRA), multi-task learning, and hyper-parameter tuning.

5.1 Curriculum Learning

Curriculum learning is a method that involves presenting the data batches to the model in order of increasing complexity. Similar to how the human brain learns in order of increasing difficulty, the training starts with simple examples. More complex examples are gradually introduced, which should help the model converge more effectively. This experiment's training data was made into simple and complex queries based on whether the correct SQL query contained concatenations or aggregations.

5.2 SQL Cross-Domain Model

Traditional methods of designing SQL generative models are typically designed with a specific task in mind or a particular dataset. For this experiment, we did not limit the number of domains and built our cross-domain dataset that was used to train and test the model (shown in the dataset section above)

5.3 Decomoposed Low-Rank Adaptation

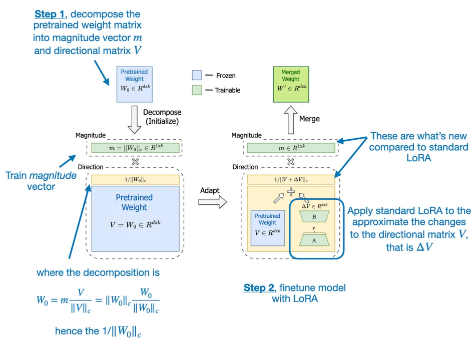
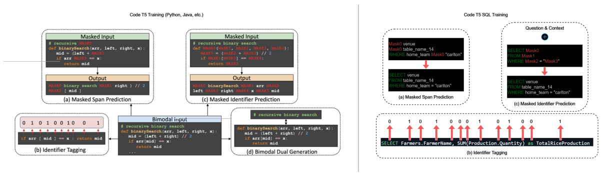

Step 1, decompose the pretrained weight matrix into magnitude vector m and directional matrix V
Pretrained Weight $W_0 \in R^{dxk}$
— Frozen
— Trainable
Merged Weight $W' \in R^{dxk}$
Decompose (Initialize)
Merge
Magnitude
$m = ||W_0||_c \in R^{1xk}$
Magnitude
$m \in R^{1xk}$
These are what's new compared to standard LoRA
Train magnitude vector
Direction
$1/||W_0||_c$
Pretrained Weight
$V = W_0 \in R^{dxk}$
Adapt
Direction
$1/||V + \Delta V||_c$
Pretrained Weight
$V \in R^{dxk}$
$\Delta V \in R^{dxk}$
B
r
A
Apply standard LoRA to the approximate the changes to the directional matrix V, that is ΔV
where the decomposition is
$W_0 = m\frac{V}{||V||_c} = ||W_0||_c\frac{W_0}{||W_0||_c}$
hence the $1/||W_0||_c$
Step 2, finetune model with LoRA

Figure 2: DoRA vs. LoRA illustration

Training LLMs is computationally expensive and sometimes impossible on consumer hardware. This limitation is because when computing backpropagation, the computer has to store multiple instances of the weight matrix (e.g., a 7-billion parameter model with Adam would need to store 3x7-billion weight values to calculate the updated weights). Low-rank adoption (LoRA) was introduced as an alternative where the weight change is instead approximated using two weight matrices multiplied together. Our experiment used a newer version of LoRA called Decomposed Low-Rank Adaptation (DoRA). DoRA involves decomposing the pretrained weight matrix of the model into two matrixes: magnitude and direction. Then, LoRA is applied to the directional matrix, and the magnitude matrix is trained separately and decomposed. Applying this to our CodeT5 model with a rank of 256 and alpha of 512, we reduced the trainable parameters by almost 90% (Appendix 2).

5.4 Multi-Task Learning


Code T5 Training (Python, Java, etc.)
Masked Input
Output
(a) Masked Span Prediction
Masked Input
Output
(c) Masked Identifier Prediction
Bimodal Input
(b) Identifier Tagging
(d) Bimodal Dual Generation
Code T5 SQL Training
Question & Context
(a) Masked Span Prediction
(c) Masked Identifier Prediction
(b) Identifier Tagging

```
# recursive MASK0
def binarySearch(arr, left, right, x):
    mid = (left + MASK1
    if arr MASK2 == x:
        return mid
```

```
MASK0 binary search MASK1 right ) // 2
MASK2 [ mid ]
```

```
# recursive binary search
def MASK0(MASK1, MASK2, MASK3, MASK4):
    MASK5 = (MASK2 + MASK3) // 2
    if MASK1[MASK5] == MASK4:
        return MASK5
```

```
MASK0 binarySearch MASK1 arr MASK2
left MASK3 right MASK4 x MASK5 mid
```

```
0 1 0 1 0 0 1 0 0 1
if arr [ mid ] == x : return mid
```

```
# recursive binary search
def binarySearch(arr, left, right, x):
    mid = (left + right) // 2
    if arr[mid] == x:
        return mid
    ...
```

```
# recursive binary search
```

```
def binarySearch(arr, left, right, x):
    mid = (left + right) // 2
    if arr[mid] == x:
        return mid
```

```
Mask0 venue
Mask0 table_name_14
WHERE home_team Mask0 "carlton"
```

```
SELECT venue
FROM table_name_14
WHERE home_team = "carlton"
```

```
SELECT Mask0
FROM Mask1
WHERE Mask2 = "Mask3"
```

```
SELECT venue
FROM table_name_14
WHERE home_team = "carlton"
```

```
0 1 0 1 0 0 0 1 0 1 0 0 1
SELECT Farmers.FarmerName, SUM(Production.Quantity) as TotalRiceProduction
```

Figure 3: SQL tasks used to train CodeT5-SQL

To design an SQL-aware encoder-decoder model, we trained on a sequence of tasks that would help the model identify SQL identifiers and how to fill in the missing information. We applied masked span prediction, identifier tagging, and masked identifier predictions to accomplish this. We randomly masked SQL spans of arbitrary lengths for the masked span prediction and had the decoder recover the original values. Next, in identifier tagging, we train only the encoder to detect identifier tags that are not operations or SQL keywords (e.g., variables or function names). Lastly, we trained the model on masked identifier prediction, feeding the model both the question context and answer and masking the identifiers in the answer; this allows the model to learn to comprehend the code

Model	Rouge1	Rouge2	RougeL	RougeLsum
T5-base	0.03361	0.01087	0.02970	0.03101
CodeT5 (w/o Training)	0.15734	0.05235	0.14548	0.14561
CodeT5-SQL (w/ Training)	0.86366	0.80194	0.85007	0.84982
CodeT5-SQL (w/ DORA)	0.82283	0.73560	0.80176	0.80171
CodeT5-SQL (Adapted w/o DORA)	0.89472	0.84101	0.88143	0.88123
CodeT5-SQL (Adapted w/ DORA)	0.85836	0.78831	0.84213	0.84228

Table 2: Table of each adapted T5 model tested with their associated Rouge Score.

semantics from the question and masked code input. Lastly, for training, we trained only on the question and context without any answers and had the model predict the appropriate SQL query.

5.5 Hyper-Parameter Tuning

Throughout the project, we have tried changing the batch size, learning rate, epochs, dropout rate, optimizer, etc. Changing the hyper-parameters didn't significantly alter our model's performance (>1%). At the end of the project, we decided to stick with an AdamW Optimization function, 64 batch size, ten epochs, a dropout rate of 0.1, and a learning rate of 5e-05. These parameters seemed to perform decently well within our test, and the rest can be seen on our GitHub in one of the configuration files.

6 Training

We trained each experiment individually and then combined it to test their effects. We trained each experiment and model combination for ten epochs (2 x runs of 5 epochs) and saved the best eval model from all runs. This best eval model was used to calculate the Rouge scores in (table x). Each experiment has a config file to set up and change parameters for each model, which can be seen on GitHub in the configs folder.

7 Metrics

We tried various metrics such as BLEU, accuracy, and Rouge scores. We found that the BLEU metric, commonly used for machine translation tasks, is inappropriate for this use case. The structure of SQL allows for lots of n-gram overlap while still producing an incorrect query. This overlap leads to BLEU scores above 95 for a model with below 30% accuracy. Accuracy is also not a good metric, as specific functionalities can be implemented differently and not be captured by this metric. Today's golden standard metric in text-to-SQL generation is human evaluation or generating results using a live database. However, due to the limited resources for our implementation, we decided to use RougeN and RougeL as our primary metrics, along with training time and RAM usage (to capture DoRA contribution).

7.1 Rouge-N & Rouge-L

In our experiment, Rouge measures how well the model-generated SQL queries match the answer SQL queries. For Rouge-N, we looked at unigrams and bigrams (N = 1 and N = 2), where N represents the number of n-grams to compare in the answer and model-generated queries. The overlapping n-grams and the total number of n-grams in the answer query are used to calculate the Rouge score. Rouge-L measures the longest common subsequence (LCS) of words that appear in both the answer and model-generated query. To calculate Rouge-L, we divide the length of the LCS by the total number of words in the answer query. Rouge-Lsum is similar to Rouge-L but does the calculations on each new line or sentence. Using various Rouge scores allows us to detect issues in SQL order, minor differences in identifiers, and length of generated queries.

8 Training Time & Ram Usage

Model	Training Time	Ram Usage
T5-base	17 min/epoch	19.0 GB
CodeT5 (w/o DoRA)	17 min/epoch	18.5 GB
CodeT5 (w/ DoRA)	18 min/epoch	14.0 GB

Table 3: Table of training time and ram usage of models.

To capture the effect of DoRA on our model, we recorded the training time and VRAM usage of each model implementation. To our surprise, the runtime for the model with DoRA appeared to be longer on average than without DoRA. After investigation, this seems to happen because of the high-rank value and alpha we pass to DoRA (rank = 256 & alpha = 2 x rank = 512). Using a high rank and alpha emphasizes that the model needs to learn a lot from the training data. We saw a decent decrease in the DoRA model's VRAM usage compared to the model without DoRA. This was expected because of the large reduction in trainable parameters that needed to be stored when backpropagation.

9 Discussion

- Overall accuracy is low for the 60M model but higher for the 200M model (possibly due to the larger model's increased capacity to learn patterns such as SQL syntax and grammar rules). Less than 10% (53/782) of complex queries were classified correctly by the 60M model. For the 200M model, 73% of complex and 82% of simple queries were classified correctly.

- One frequent cause of errors in the 60M was that it frequently prefixed column names in the SQL predictions with a table name alias (e.g., t1.column_name). Most of what it learned to classify correctly were simple outputs.

- Some frequent errors in the 200M model are due to a swapped order of AND conditions or the inclusion/exclusion of quotes, which would still produce correct output. This problem of swapping motivates using execution match as a metric, which checks for identical outputs after executing the query regardless of column order or data type.

- The curriculum learning method performed worse than introducing everything at once. This result could be attributed to most data samples being simple queries. Most of the data was trained in the first run with simple queries, and only a small portion of the remaining data was added when training with the remaining harder queries. It may be better to split the dataset into more difficulty levels and more carefully chosen criteria to generate more evenly sized buckets.

- Implementing DoRA did not reduce our training time but did reduce our number of parameters and RAM usage. The most likely reason for this is the high rank and alpha values that we set, which indicate to DoRA that the model should learn a lot from the training data as this is an untrained model when we apply DoRA.

- Training on the sequence of tasks had the biggest impact on our model results. This indicates that an identifier-aware text-to-SQL model is possible and could be a good way to train future models in the field.

Our project also has some limitations due to limited resources. In this experiment, our dataset was small enough that the model frequently overfit. We could only work with the 220M parameter CodeT5 model, as that was the biggest model that would efficiently train on our system. We only had time to fine-tune the CodeT5 model architecture, but we would like to expand our methods to other architectures in the future. Lastly, which metrics to use in evaluating text-to-SQL models is still a problem in the industry, and no metric currently exists to capture and promote models to develop alternative SQL queries as an answer.

10 Conclusion & Next Steps

This paper presented a novel approach to developing a model to generate SQL queries given natural language prompts by designing a SQL context-aware transformer model. Despite our limitations in hardware and time, using techniques such as curriculum learning, cross-domain datasets, Decomposed Low-Rank Adaptation (DoRA), multi-task learning, and hyperparameter tuning, our study provided promising results. However, despite the results, further exploration is still needed in our project and the domain itself before any model can be used in a production environment. Some future plans for our project include:

- Adding more evaluation metrics, such as the SQL-Eval, where the generated query runs on a local Postgres container, and the returned result is tested for accuracy or rouge.

- Adding beam search into our model to filter out any inappropriate SQL queries (similar to the PICARD architecture)

- Adding an LSTM to our model's head to fix grammatical errors. (Noticed a few extra commas or letters in some words)

References

b mc2. 2023. sql-create-context dataset. This dataset was created by modifying data from the following sources: zhongSeq2SQL2017 & yu2018spider.

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-sql: A survey of what we have and what we expect.

Hugging Face. 2024. Clinton/text-to-sql-v1 datasets. Accessed: April 14, 2024.

Shih-Yang Liu, Chien-Yi Wang, Hongxu Yin, Pavlo Molchanov, Yu-Chiang Frank Wang, Kwang-Ting Cheng, and Min-Hung Chen. 2024. Dora: Weight-decomposed low-rank adaptation.

Yev Meyer, Marjan Emadi, Dhruv Nathawani, Lipika Ramaswamy, Kendrick Boyd, Maarten Van Segbroeck, Matthew Grossman, Piotr Mlocek, and Drew Newberry. 2024. Synthetic-Text-To-SQL: A synthetic dataset for training language models to generate sql queries from natural language prompts.

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

Xin Wang, Yudong Chen, and Wenwu Zhu. 2021a. A survey on curriculum learning.

Yue Wang, Weishi Wang, Shafiq Joty, and Steven C. H. Hoi. 2021b. Codet5: Identifier-aware unified pre-trained encoder-decoder models for code understanding and generation.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning.

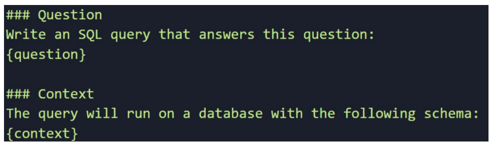
11 Appendix

A Exact String Match

Model	Exact String Match
CodeT5 60M	28.6%
CodeT5 200M	81.6%
CodeT5 200M (Simple then Complex)	75.7%

Table 4: Table of Exact String Match Results

B Prompt



```
### Question
Write an SQL query that answers this question:
{question}

### Context
The query will run on a database with the following schema:
{context}
```

Figure 4: prompt used to train model