Towards Automatic Text-to-SQL using Large Language Models
CS769 Fall 2023

Anmol Sharma
sharma265@wisc.edu

Harshit Sharma
hsharma27@wisc.edu

Harsh Sahu
hsahu@wisc.edu

Abstract

Natural language to SQL conversion is an important problem in the field of NLP and AI. Our project focuses on improving the accuracy of natural language to SQL conversion systems by fine-tuning a large language model. By providing the model with a natural language query and corresponding database schema, we aim to generate precise SQL queries. Our efforts will aid in bridging the gap between natural language and SQL and contribute to the advancement of NLP and AI.

In this project, we have fine-tuned *T5* (Raffel et al., 2020), a transformer-based large language model developed by Google, and pre-trained on a large set of varied text datasets including Wikipedia, news articles, books, etc, to work on a bunch of NLP tasks like Summarization, Question Answering, Text completion, etc. We evaluated our method on the SPIDER dataset (Yu et al., 2019), a benchmark dataset for Text-to-SQL tasks. Fine-tuning T5 on Spider, we were able to get significant performance improvement of at least 16%, increasing from 81% to 93%. We shall further investigate other proposed methods in order to improve results for the final phase of the project.

1 Introduction

Relational databases, particularly SQL databases, are widely used by organizations to store and manage their data due to their ease of use. SQL databases allow users to communicate with databases and perform various tasks like creating, updating, and deleting databases. This makes them a popular choice for any mid to large-sized company. However, retrieving data from SQL databases requires users to be skilled enough to write SQL queries. While this task may seem simple, it can be challenging as it often involves combining multiple tables and aggregating data to obtain the required information. This complexity makes writing SQL queries a difficult task.

NL: Show the names of students who have a grade higher than 5 and have at least 2 friends.

SQL:
```
SELECT T1.name
FROM friend AS T1 JOIN highschooler AS T2
ON T1.student_id = T2.id WHERE T2.grade > 5
GROUP BY T1.student_id HAVING count(*) >= 2
```

Figure 1: An example from the Spider benchmark to illustrate the mismatch between the intent expressed in NL and the implementation details in SQL. The column 'student id' to be grouped by in the SQL query is not mentioned in the question.

Additionally, many business stakeholders may not have the necessary skills or time to write these queries themselves. As a result, it would be highly beneficial if a system could convert natural language text to SQL queries automatically. This would make data retrieval from SQL databases easier and more accessible to a broader range of users, even those without advanced SQL skills.

The task of Text2SQL involves translating a natural language question about a relational database into an SQL statement that can be used to query the database. This is a significant challenge that can greatly enhance the accessibility of relational database management systems for end-users. By using Text2SQL techniques, we can create natural language interfaces for commercial RDBMSs, enabling easier access to database information. Ultimately, this can improve the user experience and make it easier for non-technical users to interact with complex relational databases.

The development of natural language interfaces for databases has been a subject of research in both the database and natural language communities for several decades. In the 1980s, methods were proposed that used intermediate logical representation to translate natural language queries into logical queries that could then be converted into database queries (Warren and Pereira, 1982).

However, these methods still relied on manually crafted mapping rules for translation. In the early 2000s, more advanced rule-based methods were introduced. (Popescu et al., 2004) utilized an off-the-shelf natural language parser to take advantage of advances in natural language processing without the need to train a parser for a specific database. However, this approach had limited coverage and was only able to handle semantically tractable questions.

In recent times, the NLP community has been actively researching deep-learning-based methods for natural language to SQL translation. These methods utilize advanced deep learning technologies. However, one of the primary challenges in developing a DL-based Text-to-SQL method is the scarcity of training data. There have been a lot of studies proposed to counter this problem. We discuss them in detail in 'Related Work'.

2 Related Works

Over the past few years, Text2SQL has emerged as a prominent research area, capturing the interest of both the database and natural language communities. Within the database community, researchers have introduced rule-based methodologies that rely on establishing connections between natural language expressions and SQL keywords. One notable system, NaLIR (Natural Language Interface for Relational databases) (Li and Jagadish, 2014), proposes a comprehensive approach that combines syntactic and semantic parsing through the utilization of Parse Trees and Mapping and Query Trees. This system aims to bridge the gap between human language and structured database queries. An alternative approach involves the transformation of input natural language into an intermediate query language, known as OQL, which operates over an ontology (Saha et al., 2016). This ontology-based system offers advantages in exploring more extensive semantic information, such as inheritance and membership relations, that may not be captured within a traditional relational schema. Meanwhile, (Baik et al., 2019) have improved the accuracy of these mappings and the inference of join conditions by using co-occurrence information of SQL fragments from query logs. While these approaches leverage structured methods for natural language processing and promise significant benefits, they encounter challenges related to lower accuracy, particularly when faced with increasingly complex and ambiguous natural language queries.

In recent years, deep learning has proven its utility in adding value across various domains, especially in tasks that involve handling substantial amounts of data, including the complex Text2SQL challenge. Several methods (Guo et al., 2019; Huang et al., 2018; Yu et al., 2018) have been proposed to tackle this problem using deep learning approaches. In one of these approaches (Guo et al., 2019), IRNet introduces a grammar-based neural model, utilizing domain knowledge as an intermediate representation to bridge the gap between natural language (NL) and SQL. This aids in addressing the disparity between the intentions expressed in NL and the SQL implementation details, thereby mitigating natural language ambiguity. The diversity in text2SQL data pairs poses an additional challenge for accurate predictions. Another approach, PT-MAML (Huang et al., 2018), suggests leveraging inferences from multiple models, each designed to handle a specific task, rather than relying on a monolithic one-size-fits-all model, as demonstrated on the WikiSQL benchmark. While this approach enhances generalization, it encounters scalability issues as the number of models and task divisions increases. An additional issue arises from the limited availability of training data, which NSP (Iyer et al., 2017) attempts to overcome by employing an interactive learning algorithm and data augmentation techniques involving templates and paraphrasing. DBPal (Basik et al., 2018) adopts a similar data augmentation strategy but incorporates a broader range of templates and diverse paraphrasing methods.

Addressing the challenge of generalization, particularly when dealing with diverse data and establishing effective relationships between text and queries, researchers have been exploring more adaptable methods utilizing large language models. For instance, a technique employing the encoder-only transformer model BERT (Guo and Gao, 2019) suggests fine-tuning on text, questions, and schema using a pretrained BERT base model. This approach utilizes three sub-models to predict the select, where, and AGG components separately. Additionally, they have also proposed a concept of match info for matching table cells information with question string and produce a feature vector which is the same length to the question. More recently, the spotlight has shifted to encoder-decoder transformers such as T5, which demonstrate enhanced performance across various natural

language multitasks (Hazoom et al., 2021)(Li et al., 2023). In one instance (Hazoom et al., 2021), the T5 model is employed to generate a task benchmark based on Stack Exchange Data and subjected to testing. This method incorporates fine-tuning with cross-entropy loss and introduces a novel metric called partial component matching (PCM) for a more realistic assessment of conversion quality. A similar evaluation technique will be adopted in our study.

3 Proposed Methodology

4 Using pre-trained T5

The T5-base pre-trained tokenizer from Hugging Face is used to tokenize both the inputs and labels in our approach. During pre-training, the T5 tokenizer learns a fixed vocabulary of subwords, which are then encoded as integer IDs for input to the T5 model. The T5 model is an encoder-decoder architecture that employs the *SentencePiece* tokenizer.

As a first step in order to benchmark the performance, we directly used pre-trained T5 (*t5-base*) to get predictions on the SPIDER dataset.

4.1 Fine Tuning T5 with schema

All model weights were unfrozen and fine-tuned using task-specific data for a total of 10 epochs. A learning rate of 10^{-6} was carefully selected and applied using the Adam optimizer to prevent significant deviation from the pretrained weights, thus retaining the benefits of the model's extensive initial training. We have used cross-entropy loss as the cost function for this task. This fine-tuning process involved $7,000$ data points for training and 1,034 data points for evaluation. As T5 model requires the task to be inputted along with the input text, we concatenate every input query along with the task description "translate English to SQL: " as a prefix.

In this step, we also pass the schema information explicitly during this process. For each data point in the Spider training dataset, we match db_id with the corresponding db_id in *tables.json*. Then we join the column names original for each training data point. Given a natural language question Q_{nl}, and its corresponding database $D_q = (T, C)$ where $T = \{t_1, t_2, ..., t_n\}$ is the list of tables and $C = \{ct_1, ct_2, ..., ct_n\}$ is the list of columns in the corresponding database, where n refers to the total number of tables, and ct_k refers to the list of column names of k^{th} table. So, the final input that goes into the model looks something like:

input = task-prefix + $Q_{nl} + \langle schema \rangle$
$+ t_1 + t_2 + \ldots + t_n + ct_1 + \ldots + ct_n$

Here, task-prefix refers to "translate English to SQL: ". And, $\langle schema \rangle$ is a "special token" for the schema. So, everything including the natural language query, table names, and columns list is fed as input during training.

4.2 Error Analysis

We analyzed the predictions generated by the fine-tuned model and found some repeating common errors that the model was making: 1) The table names in the predicted sql queries as compared to the gold sql queries were incorrect 2) The column names in some queries were also incorrect 3) Some queries did not have proper aliasing 4) Unwanted tokens like <unk> were inserted in queries that were expected to have characters like '<' or '>' in the where clause. 5) Some correctly predicted queries included <pad> or </s> token at the beginning or the end of the query. 6) Model could not predict long queries, truncating the predictions after a certain number of characters 7) WHERE clause was wrongly predicted for complex join queries. 8) JOIN clause was wrongly predicted for some queries.

Figure 2 shows some of the instances where the above errors are made in the predictions by the model.

We will fix some of these errors in the post-processing step.

4.3 Adapters

Adapters offer a significant advantage over conventional fine-tuning techniques when adapting the T5 model for NLP-to-SQL tasks. Unlike traditional fine-tuning where the entire model's weights are adjusted, adapters only modify a small subset of parameters, thereby reducing the risk of overfitting, especially when training data is limited. This parsimonious updating preserves the pre-trained knowledge of the model while endowing it with the specialized ability to translate natural language into SQL queries. Additionally, adapters provide a more efficient and resource-friendly way of customizing models for specific tasks. Instead of training multiple full-sized models for each task, one can simply use different adapters on top of a single pre-trained model. This not only economizes storage but also accelerates the adaptation process.

Gold Queries	Predicted queries	Observed Error(s)
SELECT count(*) FROM AIRLINES WHERE Country = "USA"	SELECT count(*) FROM airlines WHERE country = "USA"</s>	Unwanted token </s> included at the end of query
SELECT T2.name , count(*) FROM Friend AS T1 JOIN Highschooler AS T2 ON T1.student_id = T2.id GROUP BY T1.student_id	SELECT T1.name, count(*) FROM High_Schoolers AS T	Wrong aliasing and wrong table name
SELECT cell_mobile_number FROM Students WHERE first_name = 'Timmothy' AND last_name = 'Ward'	SELECT phone_number FROM addresses WHERE first_name = "Tim	Wrong Column Name
select distinct year from cars_data where weight between 3000 and 4000;	SELECT YEAR FROM cars WHERE weight <unk> 4000 AND YEAR > 3000	<unk> inseated of '<' and wrong table name
SELECT document_id , document_name , document_description FROM Documents	SELECT document_ID, document_name, document_description FROM Document	Wrong table name
SELECT count(*) FROM CAR_MAKERS AS T1 JOIN COUNTRIES AS T2 ON T1.Country = T2.CountryId WHERE T2.CountryName = 'france';	SELECT count(*) FROM Car_Makers WHERE continent = "	Where clause wrongly predicted for join queries
SELECT COUNT(*) FROM CARS_DATA WHERE Accelerate > (SELECT Accelerate FROM CARS_DATA ORDER BY Horsepower DESC LIMIT 1);	SELECT count(*) FROM cars WHERE accelerate = "accrob	Wrong where cluase
SELECT Name FROM teacher ORDER BY Age ASC	SELECT T1.name FROM course AS T1 JOIN teacher AS T2 ON	Join query wrongly predicted
SELECT song_name FROM singer WHERE age > (SELECT avg(age) FROM singer)	SELECT name FROM singer WHERE age > (SELECT avg(	Truncated query

Figure 2: Common errors after finetuning

In the context of NLP-to-SQL, this means quicker and more robust model specialization without compromising the rich linguistic understanding that T5 inherently possesses.

5 Experiment Setting

5.1 Dataset

We opted to use SPIDER dataset (Yu et al., 2019) for our project, which is currently the benchmark dataset for Text-to-SQL tasks. It is a large-scale, complex, and cross-domain text-to-SQL dataset that was introduced by researchers at Yale University in 2019. The Spider dataset consists of 10,181 natural language questions paired with 5,693 distinct SQL queries, spanning over 200 complex SQL schemas across 138 different domains. Figure 1 shows one of the examples of Spider Dataset. The SQL queries in the dataset can involve a wide range of SQL operations, including nested queries, aggregation functions, and joins. The training set contains 8,659 examples, the development set contains 1,034 examples, and the test set contains 488 examples.

As the test set is not publicly available, we train our model on the training set and evaluated it against the development set.

The dataset is also categorized into four levels of difficulty, namely easy, medium, hard, and extra hard, based on the complexity of the SQL queries generated for each corresponding natural language query. The easy category includes queries with simple select or aggregation queries, while medium examples include a single join and an aggregation. Hard queries involve multiple join statements in the query, while extra hard queries require nested SQL queries in addition to multiple joins and aggregation functions.

5.2 Evaluation Metric

It has been quite difficult for the researchers to comprehensively compare various studies in the domain of Text2SQL, because of the varied metrics that have been used. The evaluation strategies that have historically been used can be broadly classified into 4 categories: 1) string matching, 2) parse tree matching, 3) result matching, and 4) manual matching. String matching compares the ground truth or the GOLD query with the SQL query. When comparing a generated SQL query to the corresponding gold SQL query in string matching, certain factors can lead to incorrect judgments. For instance, the order of conditions in the WHERE clause, projected columns, and aliases can all play a role. Consider the example of two queries where one contains, $Q1$ AND $Q2$, in its WHERE clause and the other, contains $Q2$ AND $Q1$, where $Q1$ is "first_name $=$ john" and $Q2$ is "age $\leq$ 32". In this case, even though the conditions are the same, the string match algorithm would still identify them as different queries due to the different order of the conditions.

Parse tree matching compares the parse trees of two queries. Although it has fewer errors than string matching, it can still be misleading. For instance, a nested query and its flattened version may be equivalent, but their parse trees would be distinct. Result matching is a method of comparing two SQL queries by executing them on the same database and comparing their results. This assumes that identical SQL queries will always produce the same results. However, there is a risk of overestimating the similarity of two different SQL queries if they happen to produce the same results by coincidence. Manual matching involves users verifying the accuracy of translation results by manually checking the execution results of SQL

queries. However, this process requires a significant amount of manual effort and may not always guarantee reliability.

For our study, we use an improved version of string matching, *COMPONENT MATCHING* which is also one of the official metrics of Spider Dataset. It is the average exact match between the predicted and ground truth values for various SQL components. These components include SELECT, WHERE, GROUP BY, ORDER BY, and AND/OR (which is the aggregator of conditions within the GROUP clause).

To evaluate each of these components, we break them down into sets of sub-components for both the predicted and ground truth values. For example, to evaluate a SELECT component such as "SELECT avg(col1), max(col2), min(col1)", we first parse and decompose it into a set of "(avg, min, col1)" and "(max, col2)", and then compare the sets from the predicted and ground truth values to determine if they match exactly.

Previous approaches have directly compared the decoded SQL with the ground truth SQL, but this method does not take into account SQL components that do not have order constraints. In our evaluation, we treat each component as a set so that variations in the order of the components do not affect the evaluation. For example, "SELECT avg(col1), min(col1), max(col2)" and "SELECT avg(col1), max(col2), min(col1)" would be treated as the same query.

To report the overall performance of our model on each component, we compute the precision score based on exact set matching.

5.3 Approach

In general, text-to-SQL conversion task can be divided into the following steps: 1) Fine-tune the selected LLM model on the training dataset for a selected number of epochs 2) Run predictions on the test data 3) Validate the results on some validation suite against the gold dataset and get the accuracy.

The following section explains our approach in detail. In order to tackle the problem, we decided to gradually build our approach in terms of the following steps:

1. Fine-tune T5 model on training data set along with corresponding database schema information and measure performance against test data.

2. Analyze model's output and markdown common errors across various queries.

3. Change the architecture of T5 to incorporate adapter, which will help to customize the model for a specific downstream task i.e. NLP-to-SQL, and then evaluate the performance.

4. Devise and apply rules that correct these predictions as a post-processing step. Finally, evaluate the outputs.

6 Results

We measured the precision scores of our model using component matching and evaluated its performance at every step of our approach. To accomplish this, we used the spider validation test suite (Zhong et al., 2020). Figure 3 shows the comparison between the precision scores for each component SELECT, WHERE, GROUP BY, ORDER BY, and AND/OR with respect to the two approaches as of now. We find that the performance of the T5-Base model without fine-tuning was very poor for all components except for AND/OR components. Baseline T5 simply fails to predict almost all the queries. This is expected as T5 is trained on large general dataset and not for a specific downstream task. Being able to produce AND/OR clause may be attributed to the fact that T5 is pre-trained on tasks related to understanding English text, which allows it to capture AND/OR clauses accurately from the natural language query.

After performing pre-processing techniques and tokenization prior to fine-tuning the T5 model, and then modifying the prompt input during fine-tuning through schema concatenation, the model's performance significantly improved, ranging between 80% to 93% for different components. Nonetheless, it was observed that the precision score for the WHERE component remained at 0%. This shows that the WHERE clause is the most difficult of all.

7 Plan of Activities

7.1 Removing Common Errors

Going over the fine-tuned model's predictions, we noticed some repeating mistakes that the model is making in its outputs. We think that some of these errors can be fixed using case-specific hard-coded rules that will also generalize well over Test data. Below are the errors and description of our strategy on how to fix them.

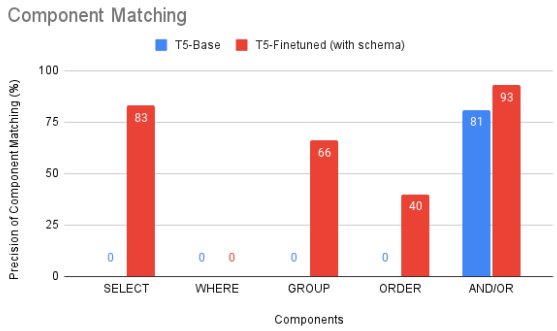

Figure 3: COMPONENT MATCHING score for each SQL component across various methods

7.1.1 Column/Table name correction

If we check Example 2 in Figure 2, the model mistakes the table name *Highschooler* as *High_schoolers*. The same error occurs in example 3 mistaking *cell_mobile_number* as *phone_number*. Below is the rule-based logic that we plan to build to correct column names:

1. Knowing the position of a column name being appeared in a query, we will extract all column names from the query. For example, all the terms mentioned after SELECT and before FROM. We will use *RegEx* for this task.

2. We will then fetch all the column names from the corresponding database schema

3. For each column name found in the query, we find its closest match in the columns extracted from the schema. We replace that column name with the match found.

We will implement the same strategy for correcting the table names

7.1.2 Table aliasing correction

One of the other commonly found errors is the wrong alias or reference of a table used in different parts of the query. If we look at example 2 in Figure 2, the model, first, assigns Table *Highschooler* as *T*, but then it mistakenly addresses its column *name* as *T1.name*. We plan to the following logic in order to fix such errors.

1. We build a dictionary of all the tables mentioned in the body of the predicted query and their assigned aliases

2. We get all the columns in the query, and find their corresponding source tables from the database schema. For columns present in multiple tables, we prefer the table for which it is the primary key

3. For each column, we then change the table alias attached as the prefix, to the correct alias of the source table

7.1.3 Replacing <unk> with >,<

We also noticed that instead of outputting the supposed '<' or '>' sign in the WHERE clause, the model predicted '<unk>' in a lot of queries. One such case is shown in example 4 of Figure 2. In order to correct this error, we needed to know the correct sign (either '<' or '>') to replace <unk> with. We will plan to come up with a generalizable strategy to mitigate this error as well.

7.2 Adding adaptors

We plan to build our custom adaptors and add them to the pre-trained T5 model. We will try different adaptor configurations in order to get the optimal results.

7.3 Building our custom Transformer

As our final effort to gain performance, we will try to significantly manipulate T5 architecture and train it on the SPIDER dataset. In this regard, we will further explore some latest research works in Text2SQL domain.

7.4 Plan of division of work

We plan to divide the future work in the following way:

- Fixing Common Errors using hard-coded rules as a post-processing step: Harsh
- Implementing Adaptors: Anmol
- Building Transformer Backed Custom Solution/Derivative: Harshit, Harsh
- Final Report writing: Anmol, Harshit, Harsh

References

Christopher Baik, Hosagrahar V Jagadish, and Yunyao Li. 2019. Bridging the semantic gap with sql query logs in natural language interfaces to databases. In *2019 IEEE 35th International Conference on Data Engineering (ICDE)*, pages 374–385. IEEE.

Fuat Basik, Benjamin Hättasch, Amir Ilkhechi, Arif Usta, Shekar Ramaswamy, Prasetya Utama, Nathaniel Weir, Carsten Binnig, and Ugur Cetintemel. 2018. Dbpal: A learned nl-interface for databases. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1765–1768.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-sql in cross-domain database with intermediate representation. *arXiv preprint arXiv:1905.08205*.

Tong Guo and Huilin Gao. 2019. Content enhanced bert-based text-to-sql generation. *CoRR*, abs/1910.07179.

Moshe Hazoom, Vibhor Malik, and Ben Bogin. 2021. Text-to-SQL in the wild: A naturally-occurring dataset based on stack exchange data. In *Proceedings of the 1st Workshop on Natural Language Processing for Programming (NLP4Prog 2021)*, pages 77–87, Online. Association for Computational Linguistics.

Po-Sen Huang, Chenglong Wang, Rishabh Singh, Wen-tau Yih, and Xiaodong He. 2018. Natural language to structured query generation via meta-learning. *arXiv preprint arXiv:1803.02400*.

Srinivasan Iyer, Ioannis Konstas, Alvin Cheung, Jayant Krishnamurthy, and Luke Zettlemoyer. 2017. Learning a neural semantic parser from user feedback. *arXiv preprint arXiv:1704.08760*.

Fei Li and Hosagrahar V Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing.

Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 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.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *The Journal of Machine Learning Research*, 21(1):5485–5551.

Diptikalyan Saha, Avrilia Floratou, Karthik Sankaranarayanan, Umar Farooq Minhas, Ashish R Mittal, and Fatma Özcan. 2016. Athena: an ontology-driven system for natural language querying over relational data stores. *Proceedings of the VLDB Endowment*, 9(12):1209–1220.

David HD Warren and Fernando CN Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *American journal of computational linguistics*, 8(3-4):110–122.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018. Typesql: Knowledge-based type-aware neural text-to-sql generation. *arXiv preprint arXiv:1804.09769*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suite. In *The 2020 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.