CIS 5220 – Final Project – Technical Report

Query Marksmen Team

April 2023

Team Members:

- Yicheng Shen; yichengs; Email: `yichengs@seas.upenn.edu`
- Zijing Wu; wuzijing; Email: `wuzijing@seas.upenn.edu`

Abstract

This project explores various approaches and deep learning models for the Text-to-SQL task, which aims to convert natural language questions into SQL queries, enabling seamless interaction between users and databases. Text-to-SQL research helps to bridge the gap between everyday users and complex database systems, empowering individuals with diverse technical expertise to harness the potential of data. We explore non-deep learning, basic deep learning, and advanced deep learning models, applying them to the Spider dataset—a diverse collection of text-SQL query pairs. We evaluate these models' performance on complex text-to-SQL tasks and analyze the associated challenges. Our advanced models leverage deep learning techniques, including customizing and fine-tuning T5 models and utilizing the GPT-3.5-turbo model with prompt engineering in a zero-shot approach. Additionally, we implement data preprocessing, L2 regularization, and efficient fine-tuning techniques to enhance model performance. Our project suggests that advanced models significantly outperform the base models, demonstrating the potential for achieving higher accuracy with more complex models and additional resources.

1 Introduction

In today's data-centric world, most applications rely heavily on databases for effective data storage, manipulation, and retrieval. SQL, being one of the most widely used languages for these purposes, is familiar to data analysts. However, a significant portion of the population lacks programming knowledge, creating a barrier for non-technical individuals who need to interact with databases. Addressing this challenge, our project focuses on the Text-to-SQL task, which aims to convert natural language queries into their SQL counterparts, facilitating seamless communication between users and databases. As the demand for

data-driven decision-making continues to rise, it becomes increasingly crucial to develop sophisticated models for Text-to-SQL tasks. This will empower users with varying levels of technical expertise to harness the potential of data, ultimately bridging the gap between everyday users and complex database systems.

Previous works in the text-to-SQL domain have primarily focused on developing various deep learning models and reinforcement learning techniques to enhance the accuracy and efficiency of converting natural language queries into SQL. These approaches have resulted in significant advancements in handling intricate and nested queries, as well as addressing unseen schemas. In this report, we present a progression of models, beginning with non-deep learning models and advancing to basic and sophisticated deep learning models, applied to the Spider dataset. Furthermore, we evaluate the performance of these models in tackling complex text-to-SQL tasks. Our findings offer insights into the challenges associated with this task, while also demonstrating the potential of leveraging deep learning techniques and adapting existing pretrained language models with appropriate modifications to excel in the text-to-SQL domain.

2 Related Works

There have been many prior works on text-to-SQL, with a range of approaches and techniques being proposed. One technique is encoding token types, which is used to improve the way keywords, entities, and numbers are represented in natural language queries for text-to-SQL tasks [Yu+18]. In this method, each word in the query is assigned a specific category or "type" based on its role, such as being a part of a database table, a column, or a number. By doing so, the model can better recognize and understand the importance of each word within the query. Most common one that is optimized for this task would be BERT and its fine-tune versions.

Others have utilized encoder-decoder type of language model, which at the time could outperform decoder only models. Building on the highly effective transformer architecture, the open-source T5 model serves as an excellent foundation for our work [Raf+20]. The text-to-text framework provided by T5 offers a versatile starting point for various natural language processing tasks. To adapt the model for our specific needs, the Simple T5 framework helps fine-tuning the T5-base model, which consists of 220 million parameters, and the T5-large model, which consists of 770 million parameters [Shi22]. The common approaches here would be to fine-tune T5 models with various techniques, such as low rank adaptation(LoRA) or Prefix Tuning.

Furthermore, our work was inspired by Eric Zhu and LlamaIndex, an open-source MLOps platform that demonstrates the potential of leveraging OpenAI's GPT for text-to-SQL generation [Liu22]. Their insightful approach and practical examples have guided our efforts to adapt the GPT model for producing effective text-to-SQL results.

Our project's contribution involves the integration of particular data preprocessing techniques and the application of select deep learning approaches,

such as L2 regularization, to enhance performance and generalization. Additionally, we carry out comprehensive evaluations of the proposed modifications and methods. The implementation of checkpoints and batch predictions also contributes to improved workflow efficiency, streamlining the process of training and fine-tuning models.

3 Dataset and Features

We utilized the Spider dataset from Yale University [Yu+19]. The Spider dataset is a comprehensive and diverse collection of text-SQL query pairs, specifically designed for training and evaluating natural language interfaces and components for database systems. The dataset can be accessed from this link.

The dataset is divided into three primary files, namely train_spider.json, train_others.json, and dev.json. The train_spider.json file contains 7,000 text-SQL pairs samples, while the train_others.json file has 1,659 text-SQL pairs samples. The dev.json file, used for validation purposes, consists of 1,034 text-SQL pairs samples. Each sample is a JSON object containing 6 fields:

1. question: The natural language text question input associated with the SQL query is represented in this field.
2. question_toks: This field contains a list of tokenized text question inputs, providing a tokenized version of the natural language questions.
3. db_id: This field represents the identifier for the affiliated database with the query.
4. query: The SQL query associated with the given text question is stored in this field. It is used as the label.
5. query_toks: This field contains a list of tokenized query inputs.
6. sql: The full SQL process for the given query is represented as an abstract syntax tree in this field. It captures the complete context and structure of the SQL query. We do not use this field.

In addition to the text-SQL pairs data, another important file in the dataset is tables.json, which contains additional information about the schema of databases. 3 important fields in this file are:

1. db_id: This field corresponds to the unique identifier for each database, enabling the connection of text-SQL pairs to their respective database schema.
2. table_names: This field comprises table names in the database, providing information on the structure of the database.
3. column_names: This field contains column names for each table in the database, further detailing the database schema.

In the available data, the primary feature is the question, which is presented as text in natural language. Additional features include information extracted from the database schema. An enhancement we performed in our preprocessing is concatenating the database schema information with the text question. The label consists of the query and the db_id, with the latter included to identify the corresponding database for the query. This allows for the calculation of execution accuracy of SQL queries.

4 Methodology

In this section, we describe our non-deep learning, basic deep learning, and more advanced deep learning methods along with our hyper-parameters.

4.1 Non-DL Model

Our non-DL approach leverages a random forest classifier, which is an ensemble learning method, to predict SQL queries from natural language text. It combines the strengths of multiple decision trees to make predictions. It includes three primary components: a HashingVectorizer, a RandomForestClassifier, and a LabelEncoder.

The HashingVectorizer is responsible for transforming text questions into numerical representations. It does so by converting the words in the text into a fixed number of features. We set this number to 2^{16}. Next, the RandomForestClassifier, comprising 60 individual decision trees, makes predictions based on the numerical feature vectors provided by the HashingVectorizer. The final result is the aggregated results of the 60 decision trees. Finally, the LabelEncoder converts the output labels into a format that can be used for training the RandomForestClassifier. It encodes the text into numerical values to train the model and later decodes the predicted numerical values back to their original textual form.

4.2 Base DL Model

Our base deep learning approach utilizes a custom RNN-based model. This model aims to leverage the sequential processing capabilities of Recurrent Neural Networks. The model is composed of an Embedding layer, LSTM layers, and a fully connected layer. The Embedding layer is responsible for converting the input tokens into dense vectors of fixed size. These dense vectors are then fed into the LSTM layers, which have the ability to capture and learn long-range dependencies within the input sequences. The fully connected layer maps the LSTM outputs to the final output dimension, which corresponds to the size of the SQL tokenizer's vocabulary. We set the embedding dimensions to 256 and we have 8 LSTM blocks. We use a batch size of 32. The dropout rate is 0.3. The model is trained for 50 epochs using the CrossEntropyLoss loss function and the Adam optimizer with a learning rate of $1e-5$.

4.3 Advanced DL Models

We experiment with many different approaches for solving the text-to-SQL task. After exploring various approaches, we ultimately decide to concentrate on customizing and fine-tuning T5 models. We take advantage of the pre-trained T5-base model, which has 220 million parameters, as well as the T5-large model, with 770 million parameters. Additionally, we experiment with utilizing the GPT-3.5-turbo model to tackle the text-to-SQL task. Owing to its high cost, we refrain from fine-tuning the model; instead, we engage in prompt engineering and assess the model using a zero-shot approach. Details of these two approaches are described in the following sections.

4.3.1 Fine-tuning T5 Models

We build a custom T5 model based on the Simple T5 framework [Shi22]. The original T5 model is a state-of-the-art pre-trained model that can be fine-tuned for a variety of tasks, such as machine translation, summarization, and question-answering. We perform extensive experiments on fine-tuning to increase the model's performance on the text-to-SQL task.

We add data preprocessing to improve performance. This additional step processes the database schema information, which includes table names and column names for each table. This information is then formatted into a concise and readable string that can be appended to each text question. The resulting combination of the text question and formatted database schema information constitutes a new input for the model. An example of such a formatted input is as follows:

> How many heads of the departments are older than 56? | department_management | department : department_id, name, creation, ranking, budget_in_billions, num_employees | head : head_id, name, born_state, age | management : department_id, head_id, temporary_acting

This format provides the model with a structured representation of the database schema and the natural language question, making it easier for the model to understand the context and generate appropriate SQL queries.

In addition to preprocessing, we add some custom implementations using the PyTorch Lightning module to improve our fine-tuning. One important modification is the customization of L2 regularization, which is added to the loss function during the training process to prevent overfitting. L2 regularization adds a penalty term to the loss function based on the squared magnitude of the model's weights. This encourages the model to learn simpler, more general patterns in the data, reducing overfitting and improving generalization. Complex models like T5, which have a large number of parameters, can easily overfit if not regularized properly. Adding the L2 regularization helps us to largely reduce validation loss.

To improve the efficiency of our fine-tuning and evaluation process, we also add checkpoints and functions to perform batch predictions. Checkpoints are

added during training, which saves the model's state at different stages of the training process. This allows for the possibility of resuming training from a saved checkpoint in case of any interruptions. Checkpointing also helps in selecting the best-performing model at different epochs. Batch predictions, on the other hand, speed up the evaluation process by enabling the model to predict multiple inputs simultaneously. By grouping the input data into smaller batches, the model can take advantage of parallel processing, leading to faster prediction times and more efficient resource utilization.

We conduct fine-tuning and evaluation of our model using an A100 GPU on Google Colab. During the fine-tuning process, the model employs cross-entropy loss, consistent with the original T5 model. However, we also provide an option to include L2 regularization for enhanced performance. For optimization, the AdamW optimizer is utilized. By default, our model employs a learning rate of $1e-4$ and a batch size of 16. A seed is set to be 42. We perform experiments with varying regularization strengths to determine the optimal configuration for improved performance and generalization. We also compare the performance of T5-base and T5-large models.

4.3.2 Zero-shot GPT-3.5-turbo

For this project, we used GPT-3.5 Turbo as the language model, and generate results by incorporating quesiton text, sql query, and related table schema content as prompt for the model input. Due to limited credit, we are only able to test out the zero-shot performance, which was to give the input once per question-to-query pairs. Even without leveraging the LLM's in-context learning or emergent ability, the results were significantly higher than the output queries of fine-tuned T5.

Noted that in order to fairly evaluate model performances, we chose to leave 'ERROR' as model outputs if the query GPT-3.5 generated could not be run by the database engine with table schemes.

5 Results

Our models are evaluated on the 1034 text-SQL pairs in dev.json of the Spider dataset. We evaluate the model's performance across various difficulty levels: easy, medium, hard, and extra hard, along with the overall performance across all levels. The results are presented in terms of execution accuracy, exact matching accuracy, and partial matching accuracy. Execution accuracy evaluates the correctness of the generated SQL query by executing it on the actual database and comparing the output with the ground truth results. Exact matching accuracy compares the generated SQL query with the ground truth query on a token-by-token basis. If the generated query is identical to the ground truth, it is considered a match. Partial matching accuracy assesses the model's performance in predicting specific components of the SQL query, such as SELECT, WHERE, GROUP BY, ORDER BY, etc.

With limited computational resources, we attain respectable accuracy using our advanced models, which significantly outperform our base models. Our findings indicate the potential for achieving considerably higher accuracy if a more complex model is employed and additional resources are allocated for training. Furthermore, we demonstrate that the specific techniques in data preprocessing, as well as deep learning approaches we incorporate into our models, enhance the performance of our models. Our implementation and results are released here.

5.1 Results of Basic Models

Both execution accuracy and exact matching accuracy for our non-DL and base DL models are exceptionally low (less than 1 percent). Our random forest model is able to learn some fundamental structures of SQL queries, such as SELECT, FROM, and WHERE. However, it struggles to predict any of the column names and predicates. Meanwhile, our RNN-based model fails to predict even an easy query after training for 50 epochs, only managing to learn the very first clause, which is SELECT. These two basic models demonstrate the immense difficulty and complexity of the text-to-SQL task. It becomes evident that the capacity of these simple models is insufficient to tackle this challenge effectively. Consequently, we shift our focus to more advanced DL models.

5.2 Results of Advanced Models

We realized that RNN may not be suitable for the job because only mimicking the query structure without "understanding" the meaning (or the probability distribution association) for text-to-SQL translation would be insufficient. For the advanced models, we want models to show interpretations of the combination of natural language questions and table schema contents. We learned that treating this problem as pure translation tasks would be insufficient, and some structural improvements for input data and loss function optimization to prevent over-fitting are necessary.

We started with the base T5 model, which is pre-trained by Google with 220 million parameters and is flexible enough to alter its neural network structures. After testing the validation loss for incremental epochs, we noticed that the validation loss is increasing, a potential sign of over-fitting. We then added an L2 regularization parameter in order to mitigate the effect, and the result was significant.

Another potential explanation for over-fitting is the size of the model parameters. We also fine-tuned the T5 large, which has 770 million parameters, to see if the problem can be solved. As we increase the number of epochs for T5 large, the validation loss would have a decreasing trend. Nonetheless, the addition of regularization term still helps the T5 large's performance.

For the zero-shot GPT3.5 model, we get decent improved results by prompt engineering. Leveraging MLOps tools including Langchain and GPT-index, we created an empty database with tables and columns based on the table schema

provided by the Spider dataset. We then feed the related question and table information as prompt and ask GPT3.5 for associated SQL.

The performance of all our advanced models is shown in Table 1. As we can see here, by incorporating table information in the source input and adding a regularization term, the performance of the T5 model increased significantly across all sections (easy, medium, hard, and extra hard). The general trend here is that as the magnitude model parameter increases (GPT3.5 has allegedly 100 billion+ parameters).

	Easy	Medium	Hard	Extra	All
	Execution Accuracy				
T5 base - no db - no reg	25.0	8.3	9.2	1.2	11.3
T5 base - with db - no reg	54.0	29.1	17.8	6.0	29.5
T5 base - with db - reg $\lambda = 1e-5$	50.8	28.7	25.3	5.4	29.7
T5 base - with db - reg $\lambda = 1e-2$	54.4	32.1	24.7	5.4	31.9
T5 large - with db - reg $\lambda = 1e-2$	62.1	37.2	28.7	7.8	37.0
GPT3.5 turbo - with db	75.3	58.5	32.4	23.6	52.7
	Exact Matching Accuracy				
T5 base - no db - no reg	23.8	7.8	8.6	1.2	10.7
T5 base - with db - no reg	54.0	27.6	17.8	3.6	28.4
T5 base - with db - reg $\lambda = 1e-5$	51.2	28.0	24.1	4.2	29.1
T5 base - with db - reg $\lambda = 1e-2$	55.6	31.4	23.6	4.8	31.6
T5 large - with db - reg $\lambda = 1e-2$	64.9	36.3	28.7	6.6	37.1
GPT3.5 turbo - with db	73.7	47.6	24.7	8.1	43.9

Table 1: Execution Accuracy (%) and Exact Matching Accuracy (%) on SQL queries in Spider dataset with different hardness levels. The "with db" parameter indicates the database schema is included in the input via our preprocessing method. The "reg" parameter indicates the strength of L2 regularization.

Table 2 shows execution accuracy, exact matching accuracy, and partial matching accuracy of the T5-large model which performs the best among all our T5 fine-tuned models. It is worth noticing that all partial matching accuracy scores are pretty high. Based on the case-by-case analysis from [Liu22], there are several reasons why partial matching percentages are high while exact matching percentages are not optimal.

First, in the spider dataset, many tables are intertwined by foreign key references. With no prior knowledge about other tables, the reference may not be fully comprehended by T5. For example, when the gold standard SQL chooses to not include the foreign reference column based on human judgment, T5 may choose to have it in the 'SELECT' clause because it may find the information relative.

Second, the problem of 'hallucination' created by language models also seems to occur for table and column alias. Rather than using plain table and column names, T5 sometimes will give an alias for tables and columns which made the exact match unreachable. We speculate this behavior may be due to the training

corpus of pre-trained T5.

	Easy	Medium	Hard	Extra	All
Count	248	446	174	166	1034
	Execution Accuracy				
	62.1	37.2	28.7	7.8	37.0
	Exact Matching Accuracy				
	64.9	36.3	28.7	6.6	37.1
	Partial Matching Accuracy				
SELECT	92.1	90.5	94.8	85.7	91.3
SELECT (no aggregation)	95.2	92.5	94.8	85.7	93.3
WHERE	90.6	76.0	72.4	70.0	79.8
WHERE (no operators)	92.2	84.0	79.3	80.0	85.6
GROUP BY (no HAVING)	83.3	67.9	93.1	53.3	73.4
GROUP BY	77.8	66.7	93.1	53.3	72.0
ORDER BY	64.0	82.6	93.3	68.8	79.5

Table 2: Accuracy (%) of our T5-large model.

Figure 1 presents the training and validation loss curves during the fine-tuning process of the T5-large model. A noticeable increase in validation loss beyond Epoch 4 indicates potential overfitting. Consequently, we select the model from Epoch 4 as the optimal choice. According to our evaluation, the Epoch 4 model outperforms those from other epochs in terms of accuracy.

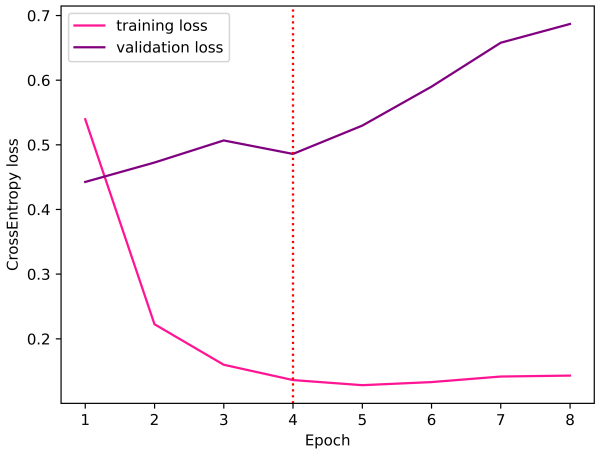

Figure 1: Loss in fine-tuning the T5-large model.

6 Discussion

We tested out various machine learning and deep learning structures, and found that neural networks, with transformer architecture, large amount of parameters, and fine-tuned on domain knowledge, could generate the best results for text-to-SQL taska.

6.1 Findings

The findings of this study reveal interesting insights into the performance of various models used in the research. A comparison of the non-deep learning (non-DL), base deep learning (base-DL), and advanced models indicates that the advanced model outperformed the other models in terms of accuracy and robustness. The non-DL model, while simple and interpretable, showed limitations in capturing complex patterns in the data, resulting in lower accuracy. The base-DL model, though more complex, did not perform as well as the advanced model due to its limited capacity to handle the intricacies of the data. In contrast, the advanced model, with its sophisticated architecture and advanced features, exhibited superior performance in terms of accuracy and generalization, indicating its effectiveness in addressing the research problem. The addition of L2 regularization for performance improvement further showed that fine-tuning techniques still stays relevant in the LLM world.

However, we do see that generic vanilla large language model can outperform smaller models by large percentages. The force brought by the scale of the model is still tremendous. Further research direction could incorporate different levels of fine-tuning on LLMs to get the optimal results.

6.2 Limitations and Ethical Considerations

Despite the promising results, this study has certain limitations. One limitation is that the advanced model may still face challenges in generalizing to unseen data, as the data used for training and testing may not fully represent the real-world scenarios. For corporate business intelligence where there would be at least thousands of tables and dominantly complex queries, the model performance could be degraded. Additionally, the advanced model, especially GPT 3.5's prompt engineering style, may also be susceptible to adversarial attacks, where malicious inputs could potentially deceive the model and lead to inaccurate predictions. Furthermore, ethical considerations should also be taken into account, as the data used for training and testing may contain biases that could affect the fairness and reliability of the model's predictions. Last but not least, our team is not fully aware about the training corpus of the several language models. It may be the case that GPT3.5 and T5 both have used the Spider dataset for training and that's the reason why we see such high scores.

6.3 Future Research Directions

There are two main directions for the next steps. The first direction is to use larger models. T5 has a 7 Billion parameters option, and GPT-4 could be way bigger than that. Due to the limited credit and constrained environment, we are not able to use the state-of-art models, and that would be the natural next step to embark on.

Furthermore, another direction would be to use more fine-tune mechanisms. Besides LoRA and prefix tuning, prompt engineering and few-shot chain of thoughts(COT) iterations would potentially greatly enhance the performance results. This comes down to funding as well, because for zero-shot, the queries are generated right away. For the chain of thoughts, it may take several iterations to get the results, and the cost may increase in magnitude if we are charged by the input tokens and generated tokens. Based on the back-of-envelop estimation, if we use GPT-4 with few shots optimization, it may cost approximately 200 to 500 times more for a single test, depending on how long the logic chain would be implemented.

7 Conclusions

In conclusion, our project investigates a range of techniques, including non-deep learning, basic deep learning, and advanced deep learning methods, for the Text-to-SQL task. We employed a random forest classifier for our non-DL approach, a custom RNN-based model for our basic DL approach, and customized and fine-tuned T5 models for our advanced DL approach. To enhance model performance, we incorporated various strategies such as data preprocessing, prompt engineering, and L2 regularization. Additionally, we integrated checkpoints and batch predictions to improve efficiency during training and evaluation. Our comprehensive methodology addresses the Text-to-SQL task effectively, and the results from our experiments have been presented and discussed throughout the project. With the evaluation of the performance of the GPT-3.5-turbo model, we demonstrated the significant impact of large language models on Text-to-SQL tasks and business intelligence. However, we also emphasized the continued importance of domain knowledge and relevant data for further optimization and improvement.

8 References

References

[Liu22] Jerry Liu. *LlamaIndex*. Nov. 2022. DOI: 10.5281/zenodo.1234. URL: https://github.com/jerryjliu/llama_index.

[Raf+20] Colin Raffel et al. *Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer*. 2020. arXiv: 1910.10683 [cs.LG].

[Shi22] Shivanandroy. 2022. URL: https://github.com/Shivanandroy/simpleT5.

[Yu+18] Tao Yu et al. "TypeSQL: Knowledge-Based Type-Aware Neural Text-to-SQL Generation". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 2 (Short Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, June 2018, pp. 588–594. DOI: 10.18653/v1/N18-2093. URL: https://aclanthology.org/N18-2093.

[Yu+19] Tao Yu et al. *Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task*. 2019. arXiv: 1809.08887 [cs.CL].