Text2SQL Using Small Language Models

Matt Daxner
daxner@usc.edu

Colin Li
ctli@usc.edu

Yongchen Lin
linyongc@usc.edu

Chandresh Mallick
cmallick@usc.edu

Evan Shiu
hshiu@usc.edu

Abstract

In recent years, language models have shown significant promise in the domain of translating natural language (NL) to code, a subcategory of which being the translation of NL to SQL queries. Large language models, such as GPT-4, achieve high accuracy on popular Text-to-SQL datasets. However, there are many applications for Text-to-SQL with privacy and compute constraints, which could be solved by using small language models (SLMs).

In this paper, we propose an approach to fine-tuning small-parameter language models (SLMs) using supervised learning in an attempt to outperform larger general-purpose models in the Text-to-SQL task. Our approach involves complimenting existing Text-to-SQL datasets with Chain of Thought reasoning steps required to arrive at a final SQL query from natural language input. SLMs are fine-tuned on this modified dataset in order to learn better from the extra signal provided by reasoning chains. Furthermore, we employed question representation, example selection, and example organization techniques within In-Context Learning for effective prompt engineering. The results demonstrated that this approach could be effective for the selected small language model (CodeS-7b), given sufficient compute resources and more data. Further work could potentially enable it to outperform baseline models on such a task.

1 Introduction

Large Language Models (LLMs) have made significant strides in generating structured outputs, including code generation and Text-to-SQL translation. However, there are issues of privacy and cost when making API calls to LLM providers. This is especially pertinent in the Text-to-SQL domain where Language Models are often exposed to sensitive corporate data. Research is emerging in Small Language Models (SLMs), which we define as models with less than 10 billion parameters. These models require fewer computing resources and can be run locally. However, they lack the capacity of LLMs to generalize effectively and perform well on complex structured text tasks due to their smaller parameter size and limited pretraining. As a result, GPT-4 with prompt engineering, alongside other fine-tuned open-source Large Language Models, remain the most effective solution for addressing the Text-to-SQL problem.

In this project, we attempt to address this issue by fine-tuning SLMs for this task with a Text-to-SQL dataset, and complimenting this dataset with generated Chain-of-Thought reasoning steps. The fine-tuned model will be evaluated and compared to the baseline models using execution match with the ground truth SQL queries from the Spider 1.0 dataset.

We propose the following research question: Can the Text-to-SQL capabilities of SLMs be enhanced through training on reasoning chains?

2 Task Definition

Text-to-SQL is a task in natural language processing with the purpose of translating a natural language question into a SQL query so that it can be executed by a relational database. Table 1 shows an example of a natural language input, the chain of thought reasoning and the expected SQL query output. This demonstrates how we can augment standard text-to-SQL examples from existing datasets to create new examples with intermediate reasoning steps.

3 Related Work

Existing approaches for Text-to-SQL can be broadly categorized as in-context learning and fine-tuning approaches.

Input	Find the total budgets of the Marketing or Finance department.
Output	1. **Sequential Structure:** Begin with the `SELECT` clause to specify the required field. Since the question asks for "total budgets" an aggregation function (`SUM`) will be used on the `department.budget` field. 2. **Conditional Structure** Apply a `WHERE` clause to filter for the specific departments mentioned, i.e., "Marketing" or "Finance." 3. **Join Structure:** No `JOIN` is needed here, as the query only involves the `department` table. 4. **Aggregation Structure:** Use `SUM` to aggregate the budget values for the specified departments, providing the total budget for each. `SELECT SUM(department.budget) FROM department WHERE department.dept_name = 'Marketing' OR department.dept_name = 'Finance';`

Table 1: Example input and output for the Text-to-SQL task, taken from the Spider dataset (Yu et al., 2018) and altered with synthetically generated Chain of Thought reasoning.

3.1 In-Context Learning for Text-to-SQL

State-of-the-art in Text-to-SQL uses in-context learning methods with LLMs (Gao et al., 2024).

The DIN-SQL approach divided the Text-to-SQL task into 4 intermediate tasks: schema linking, question classification, different prompts for each class, and query correction module (Pourreza and Rafiei, 2023). The proposed method was evaluated on the Spider and BIRD datasets. However, the paper did not study the effect of hallucinations during the self-correction guideline generation.

Gao et al. (2024) created a novel technique for example selection and organization for few-shot prompting. This technique, called DAIL, excludes database schemas and seeks to find the optimal balance between high quality (full information), and example quantity (SQL only).

The MAGIC framework uses a self-correction multi-agent method to improve the performance of few-shot prompting (Askari et al., 2024). The authors divide their agentic framework into 3 parts: Feedback Agent (critiques generated SQL), Correction Agent (corrects generated SQL agent based on feedback), and Manager (enables coordination of this agentic framework).

3.2 Fine-tuning Small Language Models for Text-to-SQL

Wang et al. (2024) discuss the demand for Text-to-SQL capabilities without relying on LLMs. Situations like data privacy, cost of prompting and limited computational resources necessitate the development of smaller, fine-tuned models that can perform well in Text-to-SQL tasks.

Li et al. (2024) developed the CodeS model from Starcoder (Li et al., 2023b). The CodeS family of models includes 1B, 3B, 7B, and 15B parameter models. CodeS was pre-trained on corpus of natural language, natural language to code, and SQL-related data. Then, CodeS was fine-tuned on Text-to-SQL labeled data. CodeS also includes a schema filtering module.

Another model is SQLCoder[1], a 7B parameter model which was fine-tuned from Starcoder. First, SQLCoder was trained on easy and medium questions, and then on hard and extra hard questions. Later, the model was enhanced by increasing the syntactic diversity of the questions, normalizing whitespace in SQL queries and adding more training data based on failure in SQL query generation[2].

3.3 Reasoning Chains

Wei et al. (2023) demonstrated that Chain-of-Thought prompting improves performance of LLMs on reasoning tasks. Previous work on Chain-of-Thought reasoning for Text-to-SQL utilized prompting methods (Zhang et al., 2023).

Magister et al. (2023) explores the transfer of reasoning capabilities to smaller models via knowledge distillation. This is done by finetuning a stu-

[1] https://defog.ai/blog/open-sourcing-sqlcoder

[2] https://defog.ai/blog/sqlcoder2-technical-details

dent model on the Chain of Thought (CoT) outputs generated by a larger teacher model. Experiments showed that the proposed method improved task performance across arithmetic, commonsense and symbolic reasoning datasets.

Similarly, Ho et al. (2023) proposed a "Fine-tune-CoT" method that generates reasoning examples from very large teacher models to finetune on smaller models. This method used multiple different reasoning examples generated from each original training sample. The experiments were conducted on arithmetic, symbolic and commonsense reasoning datasets.

Zelikman et al. (2022) developed the "Self-Taught Reasoner" (STaR) technique for fine-tuning a language model on chain-of-thought rationale. They used GPT-J to generate reasoning chains on commonsense question-answering and arithmetic tasks, and then fine-tuned the model on those reasoning chains.

4 Hypothesis

We hypothesize that a small language model fine-tuned using chain-of-thought (CoT) reasoning will outperform its baseline model on the Text-to-SQL task.

5 Methodology

5.1 Baseline Models

We compare our fine-tuned small language model to the pre-trained version of CodeS and SQLCoder. We define small language models to be less than 10 billion parameters, so we specifically use CodeS-7B-Spider and SQLCoder-7B-2 as our baselines.

5.2 Datasets

We use the Spider dataset (Yu et al., 2018). The Spider dataset contains 10,181 questions and 5,693 unique complex SQL queries. It is split into 8,659 training samples, 1,034 development samples, and 2,147 test samples. We reduce the Spider dataset from 4 difficulty categories to 3 by combining the "hard" and "extra hard" queries. The distinction between them was not clearly defined in the dataset and difficult to differentiate.

5.3 Synthetic Data Generation and Evaluation

Existing datasets do not have reasoning chains included with the SQL query, so we use an LLM to generate synthetic data from the Spider training set. We base our process and build upon on the DIN-SQL[3] technique (Pourreza and Rafiei, 2023), we conduct a schema linking process and categorized user queries into three distinct classes: EASY, NON-NESTED, and NESTED. These classes correspond to query difficulty levels, representing easy, medium, and hard queries, respectively. Schema linking is essential because user questions typically do not require the entire database schema to generate an SQL query. Filtering out the relevant schema prior to SQL generation is critical, as including irrelevant schema introduces unnecessary information which can complicate the query generation process. Furthermore, using distinct prompts and varying the examples provided for the different difficulties can improve the generation of reasoning chains and corresponding SQL. Both of these techniques were used by DIN-SQL to improve performance (Pourreza and Rafiei, 2023).

We adopt the Structured CoT prompting style for our use-case (Li et al., 2023a). For coding tasks, this technique outperforms the original CoT (Wei et al., 2023). We modify this method to generate reasoning chains and SQL[4]. This method results in reasoning chains that will follow structures in SQL queries (i.e. sequential, conditional, join, and aggregation).

Once the reasoning chains and SQL are generated, we check for an execution match between the reasoning chains and the ground truth SQL. If there is an execution match, then we save the reasoning chains. Otherwise, we perform query correction.

In query correction, we first generate a critique of the reasoning chains and the corresponding SQL query. To generate the critique we use few-shot prompting given user question, relevant schema, generated reasoning chains and SQL, and ground truth SQL. We do this to avoid the generation of superficial chains. We use a Critique SQL Structured CoT prompt which biases the LLM to output meaningful critique. Once, the critique is generated, we few-shot prompt to rewrite the reasoning chains and SQL given the critique. We first correct chains generating syntactically incorrect SQL query. Then, we correct chains generating SQL with wrong schema. Finally, we try to correct chains that generate an executable SQL query but without an execution match.

Figure 1 visualizes the process we have de-

[3] https://github.com/MohammadrezaPourreza/Few-shot-NL2SQL-with-prompting/blob/main/DIN-SQL.py

[4] The prompts used are provided here.

scribed.

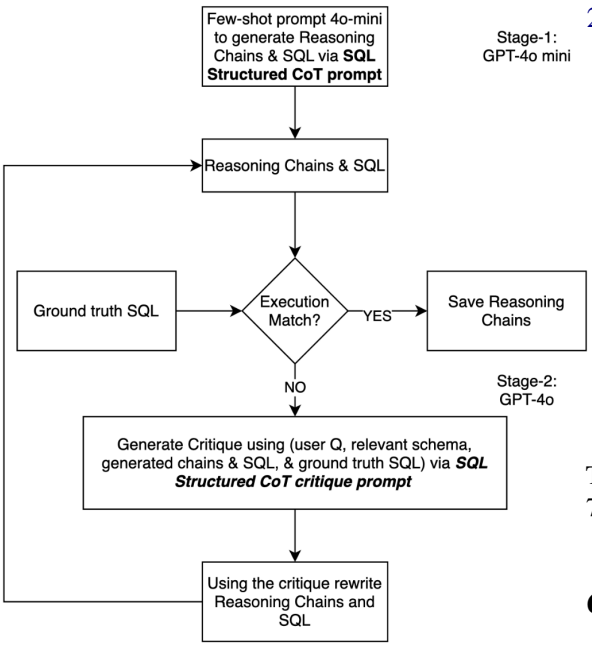

Figure 1: Flowchart of CoT Generation

Model	Easy	Med	Hard	All
SQLCoder-7B	70.5	58.6	42.2	59.8
CodeS-7B-Spider	91.4	71.1	67.5	78.4

Table 2: Evaluation of baseline models on the Spider test set.

5.4 Fine-Tuning

After we created a dataset of valid reasoning chains, we fine-tune CodeS-7B-Spider using LoRA (Hu et al., 2021) to efficiently update model weights and 4-bit quantization with BitsandBytes[5] to minimize the amount of VRAM required during training. The model was fine-tuned with the hyper-parameters in Table 3. These hyper parameters are similar hyper parameters to the parameters used by CodeS (Li et al., 2024). We fine-tune the model twice, once with a prompt designed for zero-shot prompting and a prompt for one-shot prompting[6].

During fine-tuning we first fine-tune our model on EASY queries, followed by NON-NESTED queries, and finally NESTED queries. We do so because it is empirically better to train language models on a gradual level of difficulty (Xu et al., 2020).

Hyperparameter	Value
Learning Rate	5e-5
Batch Size	16
Number of Epochs	4
LoRA Rank (r)	16
LoRA Alpha	32
LoRA Dropout	0.1
LoRA Bias	None
Quantization	4-bit
Max Tokens (Input Length)	2048

Table 3: Hyper-parameters used for fine-tuning CodeS-7B-Spider for both zero-shot and one-shot.

6 Evaluation

We use the Spider dataset to evaluate our model on Text-to-SQL. Spider has a test set with 2,147 question-SQL pairs. We compare the execution accuracy (EX%) of SQL queries generated by our model with the baselines.

We use execution accuracy instead of exact match because different SQL queries can have the same execution. So, we believe execution accuracy is a better representation of our model's performance. The exact match metric results in a much lower accuracy without properly representing the models skill at generating functional, correct queries.

Execution match is computed by testing the model's predicted queries and gold queries on SQLite databases provided by the Spider dataset. If the gold query and the predicted query returned the same result, they are said to be an execution match.

7 Experimentation and Results

We evaluate the baseline models using Spider's test set, and record the execution accuracy (EX%) results. As mentioned earlier, we reduce the Spider dataset classification to three different classes: Easy, Medium and Hard. We observe that the models performed worse on more difficult questions, as expected. Inference on the CodeS-7B-Spider and SQLCoder were done with 4-bit quantization, due to limitations of hardware we had access to. The results of the evaluation are shown in Table 2.

[5] https://github.com/bitsandbytes-foundation/bitsandbytes

[6] The prompts used for fine-tuning are provided here.

We chose to fine-tune the CodeS-7B-Spider model based on the baseline results and the fact that the model has already been fine-tuned on the Spider dataset.

7.1 Synthetic Query Generation

Initially, we select 175 queries for query generation using the prompt methodology described earlier. Synthetic generation was done using two different models (in 2 stages): GPT-4o and GPT-4o mini. We found that GPT-4o was outperforming GPT-4o-mini by only 2% in stage 1. To save cost, we opt to use GPT-4o mini for query generation. In the query critique and correction phase, we switch to GPT-4o as there was a significant performance difference in GPT-4o's query correction capabilities. We assume this is because critique generation and query correction require significant reasoning capabilities.

In general, we have two significant points of failure in synthetic data generation after the query generation. Those being:

- The generation of reasoning chains that produce syntactically incorrect SQL. This happens because the LLM picks wrong schema or adds an invalid keyword to the SQL query
- The generation of reasoning chains that produce SQL queries that do not follow the user's intent

Figure 2 shows the frequency of reasoning chains that lead to executable SQL but these do not answer the user's question.

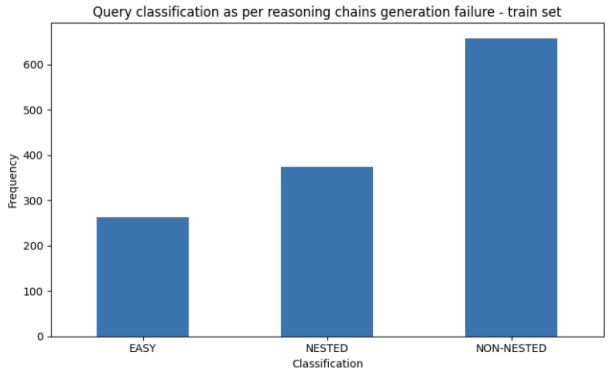

Figure 2: Chain of Thought Generation Failure

Figure 3 illustrates the frequency of query failures caused by execution errors, while Figure 4 highlights the underlying reasons for these failures.

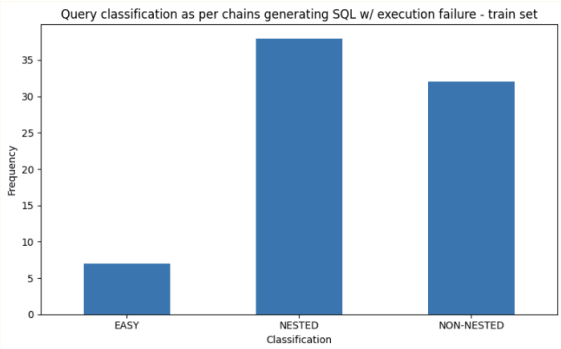

Figure 3: SQL Query Execution Failures. These are non-executable queries as generated by the reasoning chains

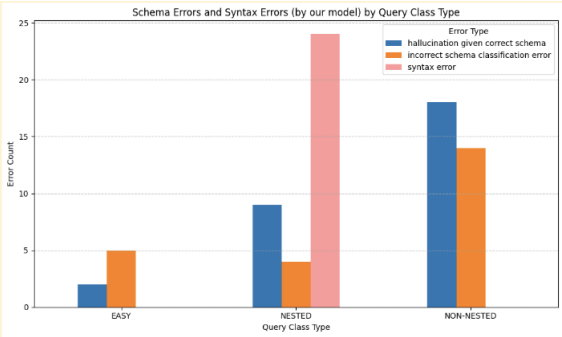

Figure 4: Schema and Syntax Errors Across Difficulties

After, correcting non-executable SQL caused due to incorrect schema and the use of wrong keywords, we get Figure 5. Figure 6 shows that there is a decrease in chain of thoughts producing SQL with execution failure.

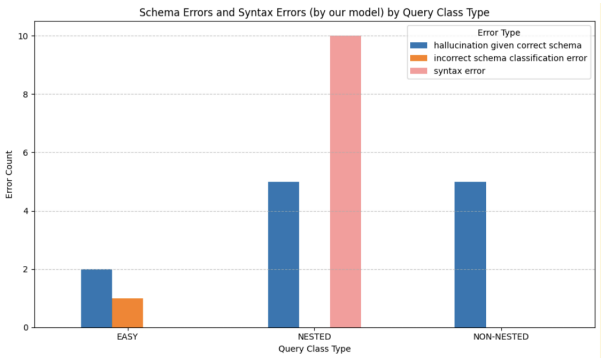

Figure 5: Schema and Syntax Errors by Difficulty After Syntax Correction

After reducing the SQL execution failure issues we fix the execution match issue. We prompt the LLM so that the number of chains generating execution match increases and Figure 8 shows that.

We are still able to improve the chain of thought generation accuracy of the train set from 85.9% to

Figure 6: Decrease in reasoning chains producing SQL Execution Failures

90.27%. 94.37% of total errors are not execution match errors - meaning chains generated a valid SQL but the SQL did not answer the user's question. This means that even after stage 2, the problem of no-execution match is still prevalent. We only ran the query correction phase once due to budget constraints. We believe running stage 2 once more would have significantly increased our synthetic data generation accuracy.

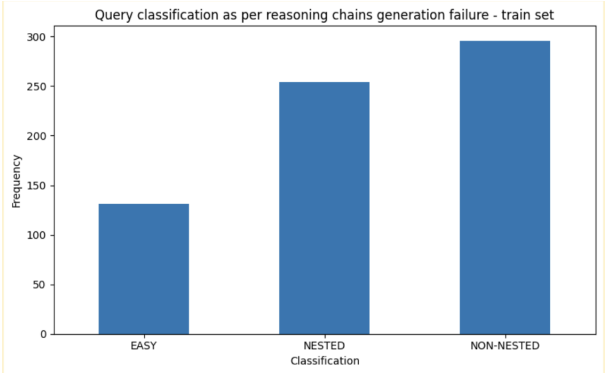

Figure 7: Increase in chains producing SQL queries with exact match

Overall, GPT-4o-mini produced non-executable SQL queries by picking wrong schema or adding incorrect keywords. To address these issues, we introduce two separate prompts to handle these issues via GPT4-o. We first improve reasoning chains, so that the SQL queries no longer contain incorrect SQL keywords. Secondly, we solve the problem of incorrect schema classification. After making sure that reasoning chains producing non-executable SQL queries went down, we solved the problem where reasoning chains were producing executable SQL but without an execution match. So to solve this we had a separate prompt.

We were able to generate reasoning chains with accuracies: 90.27% on the train set, 87.04% on the val set, and 86.39% on the test set. Across all the datasets, we noticed that the most common issue was: the chains generated executable SQL queries, but the generated SQL had no execution match with the ground truth SQL.

7.2 Model Fine-tuning

We fine-tune CodeS-7B-Spider only on reasoning chains which produced contextually and syntactically correct SQL, the rest were discarded. Over 4 epochs of training the loss continued to drop, flattening out between the 3rd and 4th epoch. The model does not seem to over-fit, as best results with the test set were seen near the end of training.

In addition, we use two different prompts for training. One prompt is designed for zero-shot training and another for one-shot training.

Figure 8 plots the loss for the model trained on zero-shot prompt. We pick the checkpoint corresponding to the 4000th step as it performed the best on validation and test set during inference.

Figure 9 plots the loss for the model trained on one-shot prompt. We pick the checkpoint corresponding to the 2000th step as it performed the best on validation and test set during inference.

7.3 Inference

Inference is conducted on the two fine tuned models multiple times at varying checkpoints(saved every 500 steps). Both models use slightly different prompts, one included an example while the other did not. We found that the one-shot model performed best at 2000th checkpoint and the zero-shot checkpoint performed best at the 4000th checkpoint. The one-shot model achieves an overall execution accuracy of 70.5% while the zero-shot model achieves an overall execution accuracy of 73.2%. The break-down of accuracies across difficulties is seen in Table 4.

Model	Easy	Med	Hard	All
0-shot	89.0	66.8	44.3	73.2
1-shot	89.7	62.6	49.4	71.2

Table 4: Execution Match of 1-shot and 0-shot models across difficulties when evaluated on the test set.

8 Conclusion

We present a SLM-based Text-to-SQL approach to improving text-to-SQL interpretability through

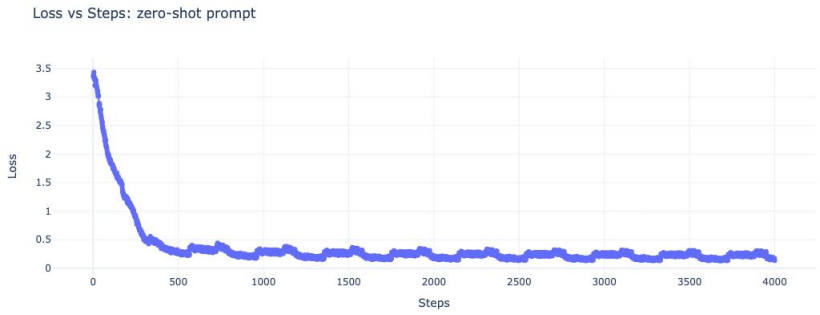

Figure 8: Loss of zero-shot Fine-tuning during fine-tuning

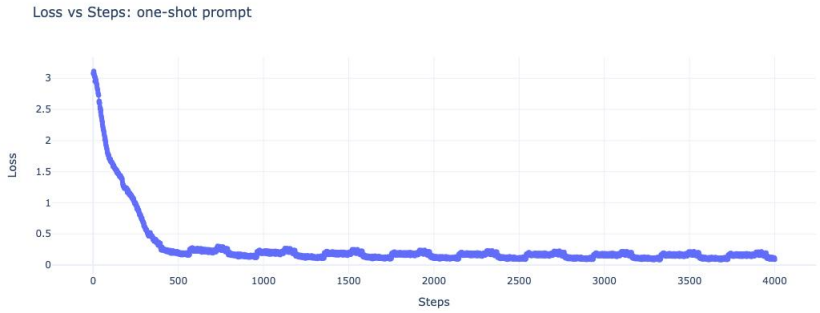

Figure 9: Loss of One-shot Fine-tuning during fine-tuning

fine-tuning smaller language models on reasoning chains. Our experiments demonstrate both the promise and limitations of this approach. We successfully generate high-quality reasoning chains across the Spider dataset, achieving 90.27% accuracy on the train set, 87.04% on the development set, and 86.39% on the test set. After synthetic data generation, we have very few non-executable SQL queries due to wrong schema classification, model hallucination, or syntactically incorrect SQL query. Most of the errors are due to reasoning chains producing an executable SQL query that does not follow the user's intent. This observation is the same across all splits of the Spider dataset. The model shows decent results on the development set with 72.6% with a zero-shot prompt and 71.8% with one-shot prompt. Performance on the test set with a zero-shot prompt is 73.2% and with a one-shot prompt is 71.2%. We continue to work on improving these accuracy figures.

Although our model underperforms the baseline, we believe that it is not far away. Despite current limitations, our work demonstrates that fine-tuning smaller models on reasoning chains is a viable path toward more interpretable and privacy-preserving Text-to-SQL systems. This problem still remains an open research problem for us.

9 Limitations and Future Work

The technical limitations of using LoRA and 4-bit quantization for fine-tuning impacted performance. We chose these techniques because it allowed us to use less powerful hardware. The initial CodeS-7B-Spider was trained with 8-bit quantization, allowing for a significant increase in precision in the model's weight representation. To mitigate this limitation, we could have explored broader hyperparameter tuning.

There are several areas of improvement future research would address. First, our prompt engineering approach could be enhanced by using sophisticated example selection methods and multi-shot learning, similar to the state of the art Text-to-SQL models, which have better generalization. Another issue was the quality of training data. Our generated reasoning chains in train set only corresponded to 90.27% execution match accuracy and contained redundant steps that affect performance.

Further work should focus on generating reasoning chains with higher accuracy. This could be achieved by running stage 2 of synthetic data generation more than once as majority of erroneous

Methods	EX%
Fine-tuning-based methods	
T5-3B + PICARD	79.3
REDSQL-3B + NatSQL	84.1
Graphix-T5-3B + PICARD	81.0
Fine-tuned SQL-PaLM	82.8
SFT Llama2-7B	77.8
SFT Llama2-13B	81.6
SFT CodeS-1B	77.9
SFT CodeS-3B	83.4
SFT CodeS-7B	85.4
SFT CodeS-15B	84.9
Prompting-based methods	
GPT-4 (few-shot)	76.8
C3 + ChatGPT	81.8
Self-Debug + Codex davinci	84.1
DIN-SQL + GPT-4	82.8
DAIL-SQL + GPT-4	83.1
SQL-PaLM (few-shot)	82.7
DAIL-SQL + GPT-4 + Self-Consistency	83.6
Ours	
SFT CoT CodeS-7B-Spider (Zero-shot) (4-bit quantized)	**72.6**
SFT CoT CodeS-7B-Spider (One-shot) (4-bit quantized)	**71.8**

Table 5: Evaluation of different models on Spider dev set, performance of other models from Li et al. (2024)

reasoning chains produce executable SQL query that do not follow user's intent.

References

Arian Askari, Christian Poelitz, and Xinye Tang. 2024. Magic: Generating self-correction guideline for in-context text-to-sql. *Preprint*, arXiv:2406.12692.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 17(5):1132–1145.

Namgyu Ho, Laura Schmid, and Se-Young Yun. 2023. Large language models are reasoning teachers. *Preprint*, arXiv:2212.10071.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2021. Lora: Low-rank adaptation of large language models. *Preprint*, arXiv:2106.09685.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. Codes: Towards building open-source language models for text-to-sql. *Proc. ACM Manag. Data*, 2(3).

Jia Li, Ge Li, Yongmin Li, and Zhi Jin. 2023a. Structured chain-of-thought prompting for code generation. *Preprint*, arXiv:2305.06599.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, Qian Liu, Evgenii Zheltonozhskii, Terry Yue Zhuo, Thomas Wang, Olivier Dehaene, Mishig Davaadorj, Joel Lamy-Poirier, João Monteiro, Oleh Shliazhko, Nicolas Gontier, Nicholas Meade, Armel Zebaze, Ming-Ho Yee, Logesh Kumar Umapathi, Jian Zhu, Benjamin Lipkin, Muhtasham Oblokulov, Zhiruo Wang, Rudra Murthy, Jason Stillerman, Siva Sankalp Patel, Dmitry Abulkhanov, Marco Zocca, Manan Dey, Zhihan Zhang, Nour Fahmy, Urvashi Bhattacharyya, Wenhao Yu, Swayam Singh, Sasha Luccioni, Paulo Villegas, Maxim Kunakov, Fedor Zhdanov, Manuel Romero, Tony Lee, Nadav Timor, Jennifer Ding, Claire Schlesinger, Hailey Schoelkopf, Jan Ebert, Tri Dao, Mayank Mishra, Alex Gu, Jennifer Robinson, Carolyn Jane Anderson, Brendan Dolan-Gavitt, Danish Contractor, Siva Reddy, Daniel Fried, Dzmitry Bahdanau, Yacine Jernite, Carlos Muñoz Ferrandis, Sean Hughes, Thomas Wolf, Arjun Guha, Leandro von Werra, and Harm de Vries. 2023b. Starcoder: may the source be with you! *Preprint*, arXiv:2305.06161.

Lucie Charlotte Magister, Jonathan Mallinson, Jakub Adamek, Eric Malmi, and Aliaksei Severyn. 2023. Teaching small language models to reason. *Preprint*, arXiv:2212.08410.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Preprint*, arXiv:2304.11015.

Fali Wang, Zhiwei Zhang, Xianren Zhang, Zongyu Wu, Tzuhao Mo, Qiuhao Lu, Wanjing Wang, Rui Li, Junjie Xu, Xianfeng Tang, Qi He, Yao Ma, Ming Huang, and Suhang Wang. 2024. A comprehensive survey of small language models in the era of large language models: Techniques, enhancements, applications, collaboration with llms, and trustworthiness. *Preprint*, arXiv:2411.03350.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc Le, and Denny Zhou. 2023. Chain-of-thought prompting elicits reasoning in large language models. *Preprint*, arXiv:2201.11903.

Benfeng Xu, Licheng Zhang, Zhendong Mao, Quan Wang, Hongtao Xie, and Yongdong Zhang. 2020. Curriculum learning for natural language understanding. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 6095–6104, Online. Association for Computational Linguistics.

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. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Preprint*, arXiv:2203.14465.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 3501–3532, Singapore. Association for Computational Linguistics.

A Video Demonstration

A brief demonstration of our project's functionality can be found at this link: `https://www.youtube.com/watch?v=coYyHFhAhrg`. The video shows part of our models inference and its execution match functionality.