STaR-SQL: Self-Taught Reasoner for Text-to-SQL

Anonymous ACL submission

Abstract

Generating step-by-step "chain-of-thought" rationales has proven effective for improving the performance of large language models on complex reasoning tasks. However, applying such techniques to structured tasks, such as text-to-SQL, remains largely unexplored. In this paper, we introduce Self-Taught Reasoner for text-to-SQL (STaR-SQL), a novel approach that reframes SQL query generation as a reasoning-driven process. Our method prompts the LLM to produce detailed reasoning steps for SQL queries and fine-tunes it on rationales that lead to correct outcomes. Unlike traditional methods, STaR-SQL dedicates additional test-time computation to reasoning, thereby positioning LLMs as spontaneous reasoners rather than mere prompt-based agents. To further scale the inference process, we incorporate an outcome-supervised reward model (ORM) as a verifier, which enhances SQL query accuracy. Experimental results on the challenging Spider benchmark demonstrate that STaR-SQL significantly improves text-to-SQL performance, achieving an execution accuracy of 86.6%. This surpasses a few-shot baseline by 31.6% and a baseline fine-tuned to predict answers directly by 18.0%. Additionally, STaR-SQL outperforms agent-like prompting methods that leverage more powerful yet closed-source models such as GPT-4. These findings underscore the potential of reasoning-augmented training for structured tasks and open the door to extending self-improving reasoning models to text-to-SQL generation and beyond.

1 Introduction

Large Language Models (LLMs) have demonstrated remarkable potential in various language tasks (Brown et al., 2020; Achiam et al., 2023), including text-to-SQL translation (Rajkumar et al., 2022; Liu et al., 2023a). Interacting with complex relational databases typically requires both programming expertise and a deep understanding of the underlying data. Text-to-SQL bridges this gap by allowing non-experts to ask questions in natural language, automatically translating them into SQL queries and returning the results (Cai et al., 2017; Xu et al., 2017; Yaghmazadeh et al., 2017).

Despite significant advancements in this field, most existing approaches primarily harness LLMs for their instruction-following capabilities, focusing on schema selection optimization and result refinement (Pourreza and Rafiei, 2024a), as illustrated in Figure 1. However, these prompts can be rigid and consume a substantial portion of the available context tokens. Smaller open-source models may also struggle to interpret and follow the carefully crafted prompts on which these methods rely. Moreover, this narrow emphasis on prompt engineering frequently overlooks the powerful reasoning capabilities inherent in LLMs (Liu et al., 2023b; Frieder et al., 2024). While these methods perform well on simple queries, they tend to falter when confronted with more complex ones (Eyal et al., 2023). This shortcoming is particularly problematic for non-experts, who may have trouble verifying whether the generated SQL queries accurately capture their original intent. Complex misalignments in SQL queries can be especially difficult for users to detect and correct.

To address these challenges, we reconceptualize text-to-SQL as a reasoning-driven process, enabling LLMs to handle complex queries by generating step-by-step rationales. This approach offers several key advantages:

- **Robustness for Complex Queries**: A step-by-step chain-of-thought reasoning method enables the model to systematically break down complex queries, handle intricate database schemas more effectively, and produce more accurate results.
- **Scalability through Reasoning**: By allocating additional computational resources at in-

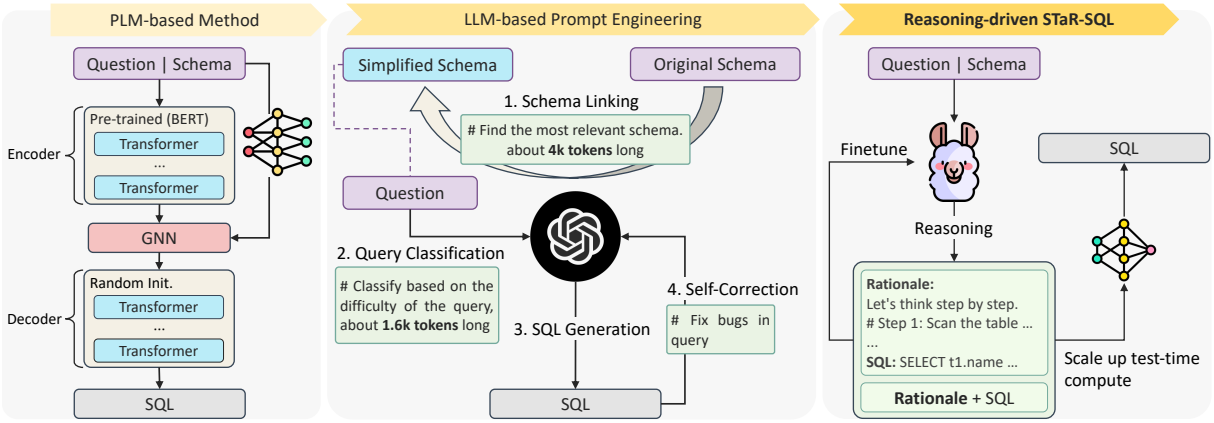

Figure 1: A comparison of different text-to-SQL methods: Traditional PLM-based methods focus on how to encode the schema (e.g., RATSQL (Wang et al., 2019)). Current LLM-based methods employ carefully designed prompts and subtask flows to simplify and understand the task, functioning in an agent-like manner and using many tokens in the prompt (e.g., DIN-SQL (Pourreza and Rafiei, 2024a)). We treat text-to-SQL as a reasoning-driven process. By leveraging the LLM's existing reasoning capabilities, we iteratively bootstrap its ability to generate high-quality rationales. In addition, by allocating more test-time computation, we further improve the reliability of the process.

ference time, reasoning performance can be improved. Techniques such as best-of-N sampling (Nakano et al., 2021; Askell et al., 2021; Cobbe et al., 2021) can further boost accuracy.

- **Enhanced Transparency**: Step-by-step rationales provide outputs that are more interpretable and verifiable compared to traditional end-to-end generation approaches.

Therefore, we introduce the Self-Taught Reasoner for text-to-SQL (STaR-SQL), a scalable bootstrapping method that enables LLMs to learn to generate high-quality rationales for text-to-SQL. Specifically, we employ few-shot prompting to have a LLM self-generate rationales and then refine its capabilities by fine-tuning on rationales that yield correct answers. To further improve performance on complex queries, we provide the correct answer to the model to guide the generation of useful rationales. These rationales are incorporated into the training data, allowing the model to learn to solve increasingly challenging queries. We repeat this procedure, using the improved model to generate subsequent training sets. Recently, some works have shown that LLMs can leverage additional test-time computation to improve their outputs (Snell et al., 2024; Brown et al., 2024; He et al., 2024). In our experiments, we introduced a verification mechanism to ensure result accuracy by employing an Outcome-supervised Reward Model (ORM) (Cobbe et al., 2021; Yu et al., 2023a), a straightforward yet effective verifier that demonstrably improves overall performance.

We demonstrate the effectiveness of our method on the challenging cross-domain benchmark Spider. Using the two official evaluation metrics (execution accuracy and exact set match accuracy (Zhong et al., 2020)), our method achieves an execution accuracy of 86.6%, outperforming both a few-shot baseline (+31.6%) and a baseline fine-tuned to predict answers directly (+18.0%). It even surpasses prompting methods (Pourreza and Rafiei, 2024a; Gao et al., 2023) that rely on more powerful closed-source models such as GPT-4, setting a new standard for reasoning-driven text-to-SQL approaches.

2 Related Work

2.1 Text-to-SQL

Text-to-SQL (Cai et al., 2017; Zelle and Mooney, 1996; Xu et al., 2017; Yu et al., 2018a; Yaghmazadeh et al., 2017), which aims to convert natural language instructions or questions into SQL queries, has drawn significant attention. Since the work of Dong and Lapata (2016), leading text-to-SQL models have adopted attention-based sequence-to-sequence architectures to translate questions and schemas into well-formed SQL queries. These models have increasingly benefited from pre-trained transformer architectures, ranging from BERT (Hwang et al., 2019; Lin et al., 2020) to larger language models such as T5 (Raffel et al., 2020) in Scholak et al. (2021), OpenAI CodeX (Chen et al., 2021), and GPT variants (Rajkumar et al., 2022; Liu and Tan, 2023; Pourreza and Rafiei, 2024a). Along with using pre-trained

models, various task-specific enhancements have been introduced, including improved schema encoding via more effective representation learning (Bogin et al., 2019) and fine-tuned attention mechanisms for sequence-to-sequence models (Wang et al., 2019). On the decoding side, some methods incorporate the syntactic structure of SQL (Hwang et al., 2019; Xu et al., 2017; Hui et al., 2021).

Recent advances in LLMs have also extended their multi-task capabilities to text-to-SQL. In zero-shot scenarios, a task-specific prompt is added before the schema and the question, guiding the LLM to generate an SQL query. Rajkumar et al. (2022); Liu et al. (2023a) showed that OpenAI CodeX can achieve 67% execution accuracy using this approach. Building on this, few-shot prompting strategies have been investigated. In particular, Pourreza and Rafiei (2024a); Liu and Tan (2023) proposed GPT-4-based DIN-SQL, which divides the problem into four subtasks (schema linking, classification, generation, and self-correction) and achieves strong performance on the Spider benchmark. However, Pourreza and Rafiei (2024a) also noted that DIN-SQL encounters difficulties when dealing with complex queries. In contrast to these approaches, our method reframes text-to-SQL as a reasoning task. By doing so, it leverages the inherent reasoning capabilities of LLMs to boost performance and facilitates the integration of additional reasoning techniques into text-to-SQL systems.

2.2 Multi-step Reasoning

Complex reasoning tasks have sparked extensive research in LLMs, which are crucial for handling challenging queries (Kaddour et al., 2023; Lightman et al., 2023; Huang et al., 2023). One prominent strategy is the Chain-of-Thought (CoT) prompting technique (Wei et al., 2022), along with its variants (Kojima et al., 2022; Wang et al., 2022; Yao et al., 2024), which decompose the reasoning process into sequential steps and systematically approach problem-solving in a human-like manner. To further enhance the accuracy of these intermediate steps, recent studies leverage extensive synthetic datasets, which are either distilled from cutting-edge models (Yu et al., 2023b; Luo et al., 2023) or composed of self-generated rationales (Zelikman et al., 2022; Yuan et al., 2023; Ni et al., 2022), to fine-tune the LLMs. Such training strategy effectively sharpens the models' ability to produce correct chain-of-thought reasoning.

Additionally, there is growing interest in test-time verification, which involves generating multiple candidate solutions and ranking them with a separate verifier (Cobbe et al., 2021; He et al., 2024) to select the most accurate one. For example, the DIVERSE framework (Li et al., 2022) employs a variety of CoT prompts together with a verifier to address reasoning challenges, while CoRe (Zhu et al., 2022) fine-tunes both the generator and verifier in a dual-process system, improving LLM performance on math word problems.

3 STaR-SQL

In this section, we introduce STaR-SQL, a method that evokes the intrinsic reasoning capabilities of LLMs to enhance performance on complex text-to-SQL tasks. We begin by describing the problem formulation (§ 3.1), followed by an explanation of how we generate step-by-step rationales (§ 3.2) for self-improvement. Finally, we outline our approach to verifier training and scaling up test-time compute to further enhance accuracy (§ 3.3). A schematic overview of the algorithm is provided in Figure 1.

3.1 Problem Formulations

The text-to-SQL task involves mapping a question $Q = (q_1, \ldots, q_m)$ and a database schema $S = \left[table_1(col_1^1 \ldots col_{c_1}^1), \ldots, table_T(col_1^T \ldots col_{c_T}^T)\right]$ to a valid SQL query $Y = (y_1, \ldots, y_n)$. Performance is typically evaluated using two metrics: 1) *exact match*, which compares the predicted query to the golden query in terms of overall structure and within each field token by token, and 2) *execution match*, which checks whether the prediction produces the same results as the golden query when executed on the database.

3.2 Self-Taught Reasoner

Self-Taught Reasoner (STaR; Zelikman et al. (2022)) is an iterative approach in which a language model improves itself using correctness feedback. We begin with a pre-trained LLM π_θ as a generator and an initial text-to-SQL dataset $\mathcal{D} = \{(Q_i, S_i, Y_i)\}_{i=1}^{D}$, where each instance comprises a question Q_i, a database schema S_i, and a corresponding golden SQL query Y_i. Our method also assumes a small prompt set $\mathcal{P}$ of examples with intermediate rationales R: $\mathcal{P} = \{(Q_i^p, S_i^p, R_i^p, Y_i^p)\}_{i=1}^{P}$, where $P \ll D$ (for instance, $P = 3$). Following the standard few-shot prompting procedure, we concatenate this prompt set $\mathcal{P}$ to each example in $\mathcal{D}$, then sample k ratio-

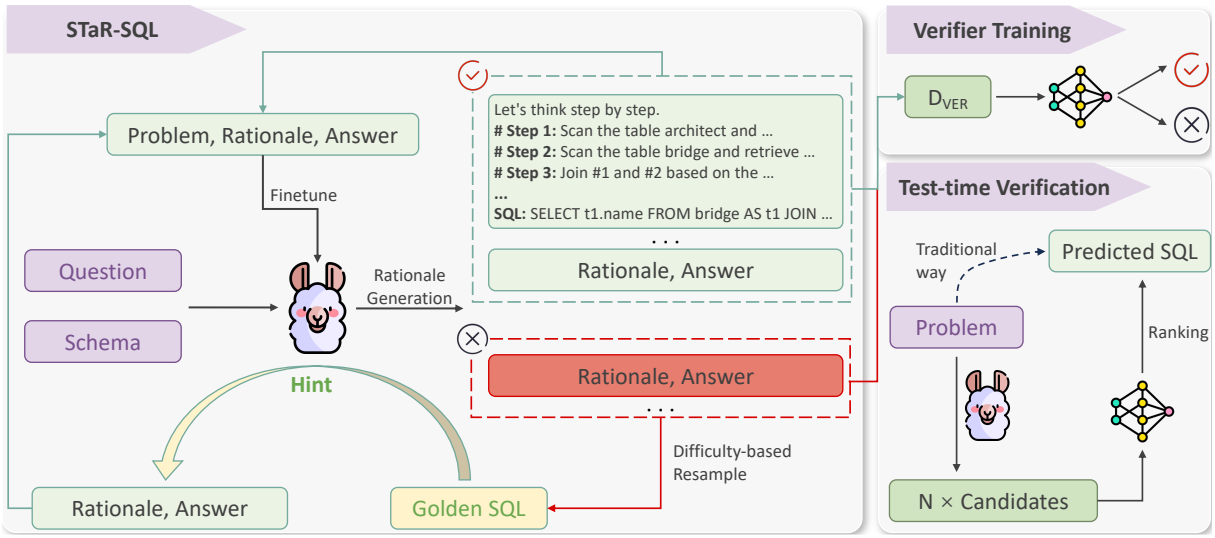

Figure 2: An overview of the STaR-SQL framework. It consists of three main steps: step-by-step rationale generation for self-improvement, verifier training, and test-time verification. We transform text-to-SQL into a reasoning task and further explore scaling up test-time computation by incorporating a verifier and employing best-of-N sampling.

nales followed by an answer from the generator: $\{(R_i^j, \hat{Y}_i^j) \sim \pi_\theta(R, \hat{Y}|\mathcal{P}, Q_i, S_i)\}_{j=1}^k$.

Having access to golden SQL queries Y_i, we can assign a binary correctness label to each generated query $\hat{Y}_i^j$ using the indicator $\mathbb{1}[\hat{Y} = Y]$. A rationale is labeled as correct if its final query $\hat{Y}$ matches the golden query Y. Intuitively, correct queries should stem from higher-quality rationales, so we only retain those correct rationales. However, under these conditions, models tend to over-sample solutions for simpler queries while under-sampling solutions for more complex queries, a phenomenon known as tail narrowing (Ding et al., 2024). This results in a training set for the next iteration dominated by rationales for simpler problems, with limited coverage of more challenging queries, thereby introducing sampling bias.

To address this issue, we employ a straightforward difficulty-based resampling strategy, which has proven sufficiently effective in practice. Specifically, for each question, we resample L times, where L is the number of incorrect initial responses for that question. To improve accuracy, we provide the golden SQL query as a hint to the model and ask it to generate rationales in the same style as during the previous rationale-generation step. Given the golden SQL query, the model can more easily reason backwards to produce a rationale that yields the correct answer. For correct initial responses, we directly add them to the training set.

We then form a new dataset, $\mathcal{D}_{\text{SFT}}$, and perform supervised fine-tuning (SFT) of the generator π_θ using the negative log-likelihood objective:

$$\mathcal{L}_{\text{SFT}} = -\mathbb{E}_{(X,R,Y)\sim\mathcal{D}_{\text{SFT}}} \sum_{i=1}^{|R|+|Y|} \log \pi_\theta(t_i|t_{<i}, X) \quad (1)$$

where X is the concatenation of the question Q and the schema S, i.e., $X = (Q, S)$.

The newly fine-tuned generator is used in subsequent iterations. Once we collect a new dataset, we always return to the *original* pre-trained model π_θ for re-initialization (as opposed to continually fine-tuning the same model) to mitigate overfitting. This process is repeated until performance plateaus.

3.3 Test-time verification

Previous self-improvement methods such as RFT (Yuan et al., 2023), STaR, and ReST (Gulcehre et al., 2023) typically discard incorrect model-generated solutions. However, even incorrect solutions can contain useful information: a language model may learn from the discrepancies between correct and incorrect solutions, identifying common error patterns and thereby improving its overall accuracy. In this work, we propose utilizing both correct and incorrect solutions in the iterative process to train a verifier. Following Cobbe et al. (2021), we introduce a verifier, also known as an outcome-supervised reward model (ORM). An ORM estimates the probability that a candidate rationale T is correct for a given problem. It is built upon a LLM with an additional randomly initialized linear layer that outputs a scalar value. The

ORM is trained with a binary classification loss:

$$\mathcal{L}_{\text{ORM}} = A_T \log r_T + (1 - A_T) \log(1 - r_T) \quad (2)$$

where A_T is the correctness label ($A_T = 1$ if T is correct, otherwise $A_T = 0$), and r_T is the ORM's sigmoid output. In our context, A_T is defined by the execution match label; i.e., whether the generated SQL query matches the golden query when executed. Since each generated rationale is labeled during every iteration, these labeled pairs form an ideal training set $\mathcal{D}_{\text{VER}}$ for the verifier.

We further scale up test-time compute through *best-of-N* sampling strategy (Nakano et al., 2021; Askell et al., 2021; Cobbe et al., 2021), which improves the reliability of the final answer. Specifically, at test time, the language model generates N candidate solutions in parallel, and the one with the highest verifier score is chosen as the final output.

4 Experiments

4.1 Experimental Setup

Datasets Several large text-to-SQL datasets have been created, some with single schemas (Wang et al., 2019) or with simple queries (Zhong et al., 2017). Notably, the Spider dataset (Yu et al., 2018b) consists of 10,181 questions and 5,693 unique complex SQL queries across 200 databases, covering 138 domains, each containing multiple tables. The standard protocol for this dataset divides it into 8,659 training examples across 146 databases and 1,034 development examples across 20 databases, with non-overlapping databases in each set. SQL queries are categorized into four difficulty levels, based on the number of SQL keywords used, the presence of nested subqueries, and the usage of column selections and aggregations. The dataset is used to assess the generalization capabilities of text-to-SQL models on complex queries with unseen schemas. We focus on this dataset for our experiments, as it enables comparison with many previous methods.

Metrics The performance of our models are evaluated using the official metrics of Spider (Zhong et al., 2020): exact-setmatch accuracy (EM) and execution accuracy (EX). The exact-set-match accuracy (EM) treats each clause as a set and compares the prediction for each clause to its corresponding clause in the reference query. A predicted SQL query is considered correct only if all of its components match the ground truth. This metric does not take values into account. The execution accuracy (EX) compares the execution output of the predicted SQL query with that of the ground truth SQL query on some database instances. Execution accuracy provides a more precise estimate of the model's performance since there may be multiple valid SQL queries for a given question, and exact set match accuracy only evaluates the predicted SQL against one of them.

Parameter Setting We used Llama-3.1-8B-Instruct as our base language model. This open-source model demonstrates non-trivial performance on the text-to-SQL task while leaving room for further improvements, making it an ideal testbed for our study. To construct the training dataset, we selected 7,000 problems from the Spider training set and sampled 8 solutions for each problem. We then filtered the correct solutions to train the generator and used the entire dataset to train the verifier. We ran STaR-SQL until performance plateaued and report the best results observed.

Baselines We conducted a comparative evaluation against several well-established methods, including traditional pre-trained transformer-based models (PLM-based) that directly predict SQL or intermediary representations. For LLM-based methods, we compared STaR-SQL with several notable prompt-engineering approaches utilizing strong closed-source LLMs, with particular emphasis on DAIL-SQL (Gao et al., 2023), which is currently the SOTA approach of this kind. We also compared our method with fine-tuned specialized code LLMs, such as CodeS (Li et al., 2024), DTS (Pourreza and Rafiei, 2024b) and ROUTE (Qin et al., 2024). Regarding training data generation, we considered Question Decomposition (QD) (Eyal et al., 2023) as a baseline. In this approach, the model is instructed to first produce a custom intermediary language, QPL, which is then translated into the rationale. To assess data quality, we compared a model trained on QD-generated data with our own approach. Finally, we included an LLM fine-tuned to predict answers directly, without revealing its reasoning steps, to demonstrate the importance of incorporating a reasoning process.

4.2 Main Results

Most of our evaluation during development was conducted on the Spider development set, which was easily accessible, unlike the test set that was only accessible through the evaluation server pro-

Classification	Methods	Models	EX	EM
PLM-based	NatSQL (Gan et al., 2021)	RAT-SQL (Wang et al., 2019)	73.7	-
	QPL (Eyal et al., 2023)	Flan-T5-XL (Chung et al., 2024)	77.4	-
	Graphix-T5 (Li et al., 2023)	Graphix-T5	78.2	**75.6**
Prompting with LLMs	Few-shot	Llama-3.1-8B-Instruct	55.0	34.2
		Qwen2.5-7B (Yang et al., 2024a)	72.5	-
		CodeX Cushman	43.1	30.9
		CodeX Davinci	61.5	50.2
		GPT-4	67.4	54.3
	DIN-SQL (Pourreza and Rafiei, 2024a)	Llama-3.1-8B-Instruct	45.2	26.5
		GPT-4	74.2	60.1
	MAC-SQL (Wang et al., 2023)	Llama-3-8B	64.3	-
		Qwen2.5-7B	71.7	-
	MCP (Qin et al., 2024)	Llama-3-8B	75.0	-
		Qwen2.5-7B	78.3	-
	DAIL-SQL (Gao et al., 2023)	GPT-3.5-TURBO	77.8	63.9
		GPT-4	81.7	69.1
Fine-Tuning with Open-Source LLMs	predict SQL-only	Llama-3.1-8B-Instruct	68.6	57.9
	QD (Eyal et al., 2023)	Llama-3.1-8B-Instruct	64.5	54.3
	CodeS (Li et al., 2024)	StarCoder	69.8	-
	DTS-SQL (Pourreza and Rafiei, 2024b)	Mistral-7B	77.1	69.3
	SENSE-7B (Yang et al., 2024b)	CodeLlama-7B	83.2	-
	ROUTE (Qin et al., 2024)	Qwen2.5-7B	<u>83.6</u>	-
	STaR-SQL	Llama-3.1-8B-Instruct	75.0	64.9
	STaR-SQL ORM@16	Llama-3.1-8B-Instruct	**86.6**	<u>72.5</u>

Table 1: Execution accuracy (EX) and exact set match accuracy (EM) (both in %) on the dev set of Spider. **Bold** indicates the best results, and <u>underline</u> indicates the second best.

vided by Yu et al. (2018b). As shown in Tables 1, our proposed method significantly enhances the original performance of Llama-3.1-8B-Instruct, improving its accuracy from 55.0% to 75.0% (+20.0%). Although small open-source models cannot directly apply reasoning to the text-to-SQL task and thus perform poorly, they demonstrate the potential to employ reasoning abilities when trained on correct rationales. Our approach also outperforms naive few-shot prompting methods, showing that it is crucial for LLMs to be familiar with the reasoning patterns required for this task: STaR-SQL surpasses few-shot prompting with stronger closed-source LLMs like GPT-4 by a large margin (+7.6%), and it is comparable to advanced prompt engineering techniques and specialized code LLMs like CodeS and DTS-SQL. Notably, it even outperforms DIN-SQL, which relies on extensive compute to simplify schemas and refine the output. Compared to predicting only the final SQL, our results demonstrate the necessity of integrating the reasoning process during inference, as this improves accuracy by an additional 6.4%.

When we scale up test-time compute, the benefit of reframing the text-to-SQL task as a reasoning process becomes even more evident. By sampling 16 solutions for each problem and applying ORM for selection, our approach significantly surpasses other PLM-based and LLM-based methods in terms of exact set match. For example, it achieves the highest accuracy of 86.6%, outperforming DAIL-SQL (the best GPT-4 prompting method) by 4.9% and the previous state-of-the-art ROUTE by 3.0%. Furthermore, training ORM does not require additional data because it is derived entirely from STaR-SQL's iterative training process. As a result, this method is both data-efficient and straightforward, leveraging both correct and incorrect solutions from an iteratively trained generator to build a robust verifier. These results highlight STaR-SQL's strong performance and scalability when increasing test-time compute.

We attribute these improvements to the following factors: 1) Reasoning Integration: Beyond leveraging the large language model's understanding capability, we also utilize its reasoning ability during inference. This transforms the model from a mere "agent" into a "reasoner," enabling it to handle complex query problems more effectively. 2) Expanded Test-Time Computation: We scale up test-time computation, which complements our approach of reframing text-to-SQL as a reason-

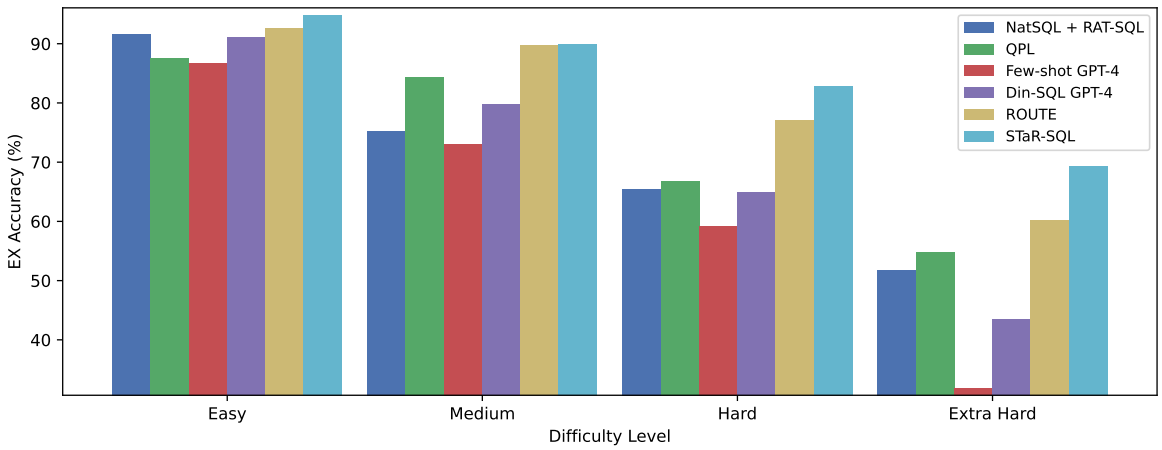

Figure 3: Execution accuracy comparison across different query difficulty levels on the Spider development set.

ing task. Allocating more computational resources proves to be an effective way to boost performance. 3) Learning from Errors: Our method also learns from the model's own erroneous reasoning rationales by using ORM as guidance. This strategy improves the accuracy of generation while maintaining data efficiency.

4.3 Execution Accuracy by Difficulty Level

We further analyzed the performance of our method on queries of varying difficulty. Figure 3 compares our approach with basic few-shot prompting and other advanced techniques on the Spider development set, demonstrating that our method consistently outperforms all baselines across every difficulty level. Although these competing methods often exceed 90% accuracy on easy queries, their performance can drop to approximately 50% on more complex ones—even specialized code LLMs fare poorly in such scenarios. This decline is particularly problematic for non-experts, who may struggle to verify whether a complex SQL query matches their intended question (Eyal et al., 2023). Notably, our method achieves the greatest gains in the extra-hard (69.3%) and hard (82.8%) categories, outperforming the second-best results by +5.8% and +9.1%, respectively. These gains stem from integrating reasoning into the inference process, leveraging the model's reasoning capabilities to address complex queries, and highlighting the importance of shifting the problem-solving paradigm.

4.4 Different Amounts of Candidate Solutions

The number of candidate solutions affects verification performance. While a larger pool of solutions

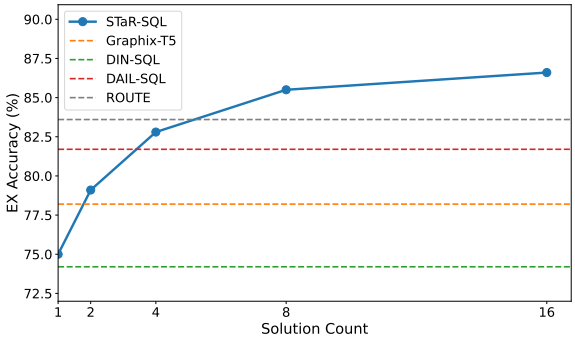

Figure 4: Performance of STaR-SQL with varying numbers of solutions (N).

can introduce additional, potentially superior candidates, it also increases computational overhead and may lead to diminishing returns. In our study, we restrict the maximum number of solutions to 16. As shown in Figure 4, increasing the number of samples consistently improves performance. Notably, sampling 4 solutions already enables STaR-SQL to surpass the best prompt-engineering method, DAIL-SQL, which depends on the more powerful but closed-source GPT-4. With 8 solutions, STaR-SQL further outperforms the state-of-the-art specialized code LLM, ROUTE, by 1.9%. These results demonstrate that substantial accuracy gains can be achieved with only a slight increase in test-time computation. Our findings align with recent work suggesting that increased test-time compute enhances reasoning performance (Snell et al., 2024; Brown et al., 2024; Wu et al., 2024). Moreover, allocating additional tokens to the reasoning process, rather than to carefully engineered prompts, proves more effective. For example, our method achieves a 41.4% improvement over DIN-SQL, which uses

more than 6k tokens in its prompt.

Example Problem Soving

Schema:
Table concert, columns = [*, concert_ID, concert_Name, Theme, Stadium_ID, Year]
Table singer, columns = [*, Singer_ID, Name, Country, Song_Name, Song_release_year, Age, Is_male]
Table singer_in_concert, columns = [*, concert_ID, Singer_ID]
Table stadium, columns = [*, Stadium_ID, Location, Name, Capacity, Highest, Lowest, Average]
Foreign_keys = [concert.Stadium_ID = stadium.Stadium_ID, singer_in_concert.Singer_ID = singer.Singer_ID, singer_in_concert.concert_ID = concert.concert_ID]
Question:
Show the stadium names without any concert.

Rationales:
#1: Scan the table concert and retrieve the stadium IDs of all concerts.
#2: Scan the table stadium and retrieve the names of all stadiums.
#3: Select the records from #2 that do not appear in #1, and identify the names of all stadiums without any concerts.

SQL: SELECT name FROM stadium WHERE stadium_id NOT IN (SELECT stadium_id FROM concert)

Figure 5: A case study from the Spider dev set.

4.5 Case Study

We also conduct a case study to intuitively demonstrate the effectiveness of STaR-SQL. As shown in Figure 5, when confronted with a complex question, STaR-SQL successfully decomposes the problem into a series of reasoning steps, progressively guiding the generation of the final SQL query. In addition, STaR-SQL enhances transparency by presenting the entire query generation process and providing a clear rationale for the final result. This transparency not only improves interpretability but also enables users to verify whether the generated query aligns with their intended question, making it easier to validate consistency between the input and output compared to other methods.

4.6 Ablation Study

We conduct an ablation study to evaluate three key components of our framework: (a) the use of intermediate rationales (step-by-step reasoning), (b) the best-of-N sampling strategy during inference, and (c) the verifier-based ranking compared to a self-consistency (majority voting) baseline. Table 2 summarizes the results under different settings. We observe that: 1) Removing step-by-step reasoning severely degrades both execution accuracy (EX) and exact match (EM), underscoring the necessity of intermediate reasoning. 2) Omitting best-of-N sampling reduces accuracy, highlighting the benefit of scaling test-time computation. 3) Replacing the verifier with self-consistency improves performance over single-shot generation but still falls short of our verifier-based approach.

Method	EX	EM
Ours	**86.6**	**72.5**
w/o rationales	68.6	57.9
w/o best-of-N	75.0	64.9
Self-Consistency	78.8	71.7

Table 2: Results of the ablation study, demonstrating the impact of different components of STaR-SQL.

5 Conclusion

In this paper, we propose STaR-SQL, an innovative method that leverages the intrinsic reasoning capabilities of language models to perform step-by-step reasoning for text-to-SQL problems. We iteratively bootstrap the ability to generate high-quality rationales and integrate a verifier to enhance the accuracy. Our empirical findings highlight the efficacy of STaR-SQL: our model achieves state-of-the-art results among fine-tuned models on the Spider dev set (without database values), especially on hard and extra-hard queries, demonstrating notable performance improvements over existing PLM-based and LLM-based methods. Through step-by-step reasoning, the large language model makes the entire process more interpretable than merely generating SQL or intermediate representations—particularly for complex queries. At the same time, by allocating additional test-time computation, we further improve accuracy, illustrating the scalability and potential of our method.

In future work, we plan to explore more effective ways of utilizing test-time compute to boost the reasoning capabilities of language models on text-to-SQL tasks. We have begun experimenting with a stronger verifier—a process-supervised reward model (PRM)—which employs fine-grained supervision signals. Beyond the best-of-N approach, there are also other methods for using test-time compute to enhance LLM performance. For instance, one can modify the proposal distribution for responses by prompting the model to sequentially revise its outputs, or alter how the verifier is used (e.g., leveraging Monte Carlo Tree Search or other search strategies). We believe these directions hold promise for further improving the robustness and accuracy of text-to-SQL systems.

Limitations

Although STaR-SQL is effective for text-to-SQL tasks under simple schema encoding, it remains uncertain whether additional methods for rich schema encoding could further enhance performance. As our approach transforms text-to-SQL into a reasoning task, we have not yet integrated techniques to improve reasoning, such as using more powerful verifiers like process-supervised reward models (PRMs) or search strategies like Monte Carlo Tree Search (MCTS). Addressing these considerations will be the focus of our future research.

Ethics Statement

The development of STaR-SQL aims to improve the accuracy and reliability of text-to-SQL tasks using Large Language Models (LLMs). While our method poses no immediate ethical concerns, we acknowledge the potential for misuse if applied in sensitive areas such as automated decision-making. We recommend rigorous evaluation and oversight to prevent bias and ensure data privacy in all applications. Transparency and adherence to ethical standards are crucial in the deployment of these technologies.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Amanda Askell, Yuntao Bai, Anna Chen, Dawn Drain, Deep Ganguli, Tom Henighan, Andy Jones, Nicholas Joseph, Benjamin Mann, Nova Dassarma, Nelson Elhage, Zac Hatfield-Dodds, Danny Hernandez, John Kernion, Kamal Ndousse, Catherine Olsson, Dario Amodei, Tom B. Brown, Jack Clark, Sam McCandlish, Christopher Olah, and Jared Kaplan. 2021. A general language assistant as a laboratory for alignment. *ArXiv*, abs/2112.00861.

Ben Bogin, Matt Gardner, and Jonathan Berant. 2019. Representing schema structure with graph neural networks for text-to-sql parsing. *arXiv preprint arXiv:1905.06241*.

Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv preprint arXiv:2407.21787*.

Tom Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. 2020. Language models are few-shot learners. *Advances in neural information processing systems*, 33:1877–1901.

Ruichu Cai, Boyan Xu, Xiaoyan Yang, Zhenjie Zhang, Zijian Li, and Zhihao Liang. 2017. An encoder-decoder framework translating natural language to database queries. *arXiv preprint arXiv:1711.06061*.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. 2021. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*.

Hyung Won Chung, Le Hou, Shayne Longpre, Barret Zoph, Yi Tay, William Fedus, Yunxuan Li, Xuezhi Wang, Mostafa Dehghani, Siddhartha Brahma, et al. 2024. Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. 2021. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*.

Yiwen Ding, Zhiheng Xi, Wei He, Zhuoyuan Li, Yitao Zhai, Xiaowei Shi, Xunliang Cai, Tao Gui, Qi Zhang, and Xuanjing Huang. 2024. Mitigating tail narrowing in llm self-improvement via socratic-guided sampling. *arXiv preprint arXiv:2411.00750*.

Li Dong and Mirella Lapata. 2016. Language to logical form with neural attention. *arXiv preprint arXiv:1601.01280*.

Ben Eyal, Amir Bachar, Ophir Haroche, Moran Mahabi, and Michael Elhadad. 2023. Semantic decomposition of question and sql for text-to-sql parsing. *arXiv preprint arXiv:2310.13575*.

Simon Frieder, Luca Pinchetti, Ryan-Rhys Griffiths, Tommaso Salvatori, Thomas Lukasiewicz, Philipp Petersen, and Julius Berner. 2024. Mathematical capabilities of chatgpt. *Advances in Neural Information Processing Systems*, 36.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R Woodward, John Drake, and Qiaofu Zhang. 2021. Natural sql: Making sql easier to infer from natural language specifications. *arXiv preprint arXiv:2109.05153*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen

Wang, Chenjie Gu, et al. 2023. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*.

Mingqian He, Yongliang Shen, Wenqi Zhang, Zeqi Tan, and Weiming Lu. 2024. Advancing process verification for large language models via tree-based preference learning. *arXiv preprint arXiv:2407.00390*.

Jie Huang, Xinyun Chen, Swaroop Mishra, Huaixiu Steven Zheng, Adams Wei Yu, Xinying Song, and Denny Zhou. 2023. Large language models cannot self-correct reasoning yet. *arXiv preprint arXiv:2310.01798*.

Binyuan Hui, Xiang Shi, Ruiying Geng, Binhua Li, Yongbin Li, Jian Sun, and Xiaodan Zhu. 2021. Improving text-to-sql with schema dependency learning. *arXiv preprint arXiv:2103.04399*.

Wonseok Hwang, Jinyeong Yim, Seunghyun Park, and Minjoon Seo. 2019. A comprehensive exploration on wikisql with table-aware word contextualization. *arXiv preprint arXiv:1902.01069*.

Jean Kaddour, Joshua Harris, Maximilian Mozes, Herbie Bradley, Roberta Raileanu, and Robert McHardy. 2023. Challenges and applications of large language models. *arXiv preprint arXiv:2307.10169*.

Takeshi Kojima, Shixiang Shane Gu, Machel Reid, Yutaka Matsuo, and Yusuke Iwasawa. 2022. Large language models are zero-shot reasoners. *Advances in neural information processing systems*, 35:22199–22213.

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. *Proceedings of the ACM on Management of Data*, 2(3):1–28.

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. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13076–13084.

Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, B. Chen, Jian-Guang Lou, and Weizhu Chen. 2022. Making language models better reasoners with step-aware verifier. In *Annual Meeting of the Association for Computational Linguistics*.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2023. Let's verify step by step. *arXiv preprint arXiv:2305.20050*.

Xi Victoria Lin, Richard Socher, and Caiming Xiong. 2020. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *arXiv preprint arXiv:2012.12627*.

Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. 2023a. A comprehensive evaluation of chatgpt's zero-shot text-to-sql capability. *arXiv preprint arXiv:2303.13547*.

Hanmeng Liu, Ruoxi Ning, Zhiyang Teng, Jian Liu, Qiji Zhou, and Yue Zhang. 2023b. Evaluating the logical reasoning ability of chatgpt and gpt-4. *arXiv preprint arXiv:2304.03439*.

Xiping Liu and Zhao Tan. 2023. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv preprint arXiv:2304.11556*.

Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jianguang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, and Dongmei Zhang. 2023. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. *arXiv preprint arXiv:2308.09583*.

Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. 2021. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*.

Ansong Ni, Jeevana Priya Inala, Chenglong Wang, Oleksandr Polozov, Christopher Meek, Dragomir Radev, and Jianfeng Gao. 2022. Learning math reasoning from self-sampled correct and partially-correct solutions. *arXiv preprint arXiv:2205.14318*.

Mohammadreza Pourreza and Davood Rafiei. 2024a. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36.

Mohammadreza Pourreza and Davood Rafiei. 2024b. Dts-sql: Decomposed text-to-sql with small large language models. *arXiv preprint arXiv:2402.01117*.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2024. Route: Robust multitask tuning and collaboration for text-to-sql. *arXiv preprint arXiv:2412.10138*.

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. *Journal of machine learning research*, 21(140):1–67.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. 2022. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. Picard: Parsing incrementally for constrained auto-regressive decoding from language models. *arXiv preprint arXiv:2109.05093*.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling llm test-time compute optimally can be more effective than scaling model parameters. *arXiv preprint arXiv:2408.03314*.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2019. Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers. *arXiv preprint arXiv:1911.04942*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

Yangzhen Wu, Zhiqing Sun, Shanda Li, Sean Welleck, and Yiming Yang. 2024. Inference scaling laws: An empirical analysis of compute-optimal inference for problem-solving with language models. *arXiv preprint arXiv:2408.00724*.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. 2017. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, et al. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024b. Synthesizing text-to-sql data from weak and strong llms. *arXiv preprint arXiv:2408.03256*.

Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2024. Tree of thoughts: Deliberate problem solving with large language models. *Advances in Neural Information Processing Systems*, 36.

Fei Yu, Anningzhe Gao, and Benyou Wang. 2023a. Outcome-supervised verifiers for planning in mathematical reasoning. *arXiv preprint arXiv:2311.09724*.

Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. 2023b. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*.

Tao Yu, Zifan Li, Zilin Zhang, Rui Zhang, and Dragomir Radev. 2018a. 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, et al. 2018b. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*.

Zheng Yuan, Hongyi Yuan, Chengpeng Li, Guanting Dong, Chuanqi Tan, and Chang Zhou. 2023. Scaling relationship on learning mathematical reasoning with large language models. *arXiv preprint arXiv:2308.01825*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. 2022. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the national conference on artificial intelligence*, pages 1050–1055.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. *arXiv preprint arXiv:2010.02840*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103*.

Xinyu Zhu, Junjie Wang, Lin Zhang, Yuxiang Zhang, Yongfeng Huang, Ruyi Gan, Jiaxing Zhang, and Yujiu Yang. 2022. Solving math word problems via cooperative reasoning induced language models. *arXiv preprint arXiv:2210.16257*.