FROM PIPELINE-BASED GENERATION TO INTERACTIVE REASONING TRAJECTORY LEARNING FOR TEXT-TO-SQL

Anonymous authors
Paper under double-blind review

ABSTRACT

In the intersection of natural language processing (NLP) and structured databases, translating natural language queries into SQL remains a formidable challenge. While current methods, which combine multi-stage pipelines with LLMs, have made substantial progress in generating executable SQL, they still encounter issues such as error propagation and high generation latency. To address these challenges, we propose STIR-SQL, a Text-to-SQL agent that performs single-trajectory interleaved reasoning and is trained via pure reinforcement learning with multi-dimension reward. Specifically, STIR-SQL replaces the multi-stage pipeline with a single, tool-integrated interleaved reasoning trajectory before the final answer, directly reducing generation latency. Within this trajectory, STIR-SQL interacts with specialized tools and performs self-correction, which mitigates error propagation and enhances the accuracy of SQL generation. Experimental results show that our approach achieves an execution accuracy of 69.7% on the BIRD benchmark, while requiring only 3B activated parameters during inference in a 30B-scale model. Furthermore, the reduction in activated parameters and inference latency demonstrates its strong potential for real-world deployment.

1 INTRODUCTION

Text-to-SQL translates natural-language questions into executable SQL, enabling non-expert users to access structured databases and supporting the democratization of data analysis Deng et al. (2022); Katsogiannis-Meimarakis & Koutrika (2023). Early systems relied on rule-based templates Zelle & Mooney (1996); Li & Jagadish (2014), while later neural semantic parsers Cao et al. (2021); Cai et al. (2021); Hui et al. (2022) improved flexibility but remained dependent on task-specific architectures and showed limited cross-domain generalization Guo et al. (2019).

The emergence of LLMs has led to three main paradigms: prompt-based methods such as DIN-SQL Pourreza & Rafiei (2023) and DAIL-SQL Gao et al. (2023); post-training methods based on supervised fine-tuning or reinforcement learning Shen et al. (2021), including SQL-R1 Ma et al. (2025), Reasoning-SQL Pourreza et al. (2025), and Arctic-Text2SQL-R1 Yao et al. (2025); and hybrid approaches such as XiYan-SQL Yao et al. (2025) that further exploit test-time scaling. Although these methods improve accuracy, efficiency remains challenging on complex databases.

Recent systems therefore commonly adopt multi-stage pipelines Maamari et al. (2024); Talaei et al. (2024). As illustrated in Fig. 1, they sequentially perform schema linking, SQL generation, and post-processing: the system searches database contents and selects relevant tables and columns, generates SQL from the selected schema, and then refines candidates through ranking or self-correction Qin et al. (2022). However, this design has two major limitations. First, schema-linking errors can distort the grounding context and propagate to the final query Liu et al. (2025a). Second, database retrieval and candidate comparison introduce substantial latency; pairwise ranking may require $O(N^2)$ LLM calls and nearly two hundred calls when $N = 16$ Shi et al. (2025).

Inspired by ReAct Yao et al. (2022), we investigate a single-trajectory interleaved reasoning paradigm that replaces separate stages with a unified "Think-Act-Observe" loop. The model reasons, interacts with database tools, observes feedback, and immediately revises its decisions within one generation trajectory. This design limits error propagation and avoids massive database vectorization and expensive candidate re-ranking, with computation growing approximately linearly with the number of reasoning turns. Its realization, however, raises three challenges.

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

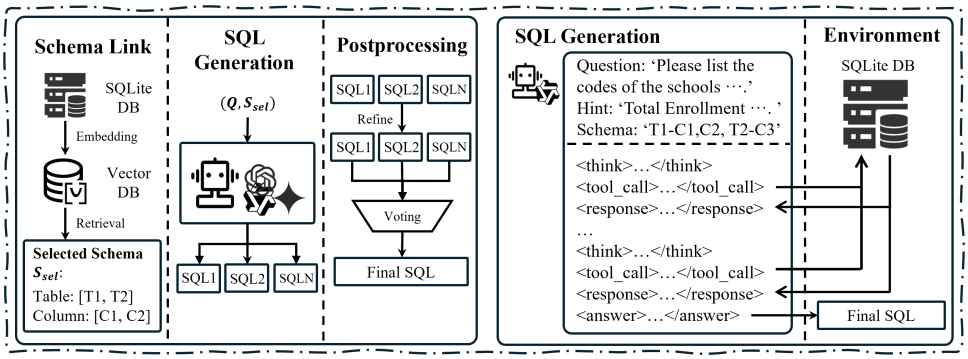

Figure 1: Comparison between multi-stage LLM-based method and single-trajectory interleaved reasoning.

Challenge 1: Semantic ambiguity. Cryptic column names and noisy values may cause incorrect schema selection unless reasoning is grounded in database facts.

Challenge 2: Balancing simplicity and expressiveness. An overly limited toolset causes inefficient trial-and-error, whereas an excessively granular action space increases decision complexity and context usage.

Challenge 3: Long-horizon reward guidance. Binary execution rewards are too sparse to distinguish near-miss SQL queries from fundamentally incorrect ones, making complex reinforcement learning difficult.

To address these challenges, we propose STIR-SQL, a Text-to-SQL agent trained through pure reinforcement learning. STIR-SQL combines descriptive database augmentation for semantic grounding, four compact tools for schema inspection, keyword checking, global keyword search, and SQL execution, and a multi-dimensional reward that evaluates schema accuracy, syntactic structure, tool compliance, and execution quality. Together, these components internalize the conventional pipeline into a unified adaptive reasoning process.

On BIRD Li et al. (2023), STIR-SQL achieves 69.7% execution accuracy with only 3B active inference parameters. Its single-trajectory design also reduces generation latency by approximately 75% relative to a multi-stage method Talaei et al. (2024). Results on Spider Yu et al. (2018) and Spider-DK Gan et al. (2021) further demonstrate strong cross-domain generalization and low inference latency.

In summary, our contributions are:

- We reformulate Text-to-SQL from pipeline-based generation into interactive reasoning trajectory learning, which unifies schema linking, SQL generation, and post-processing within a single adaptive trajectory.
- We develop STIR-SQL with four complementary tools and multi-dimension rewards, enabling reinforcement learning to utilize database feedback and optimize SQL construction throughout the reasoning trajectory.
- Extensive Experiments demonstrate the effectiveness, efficiency, and robustness of the proposed method.

2 Related Work

Text-to-SQL has evolved from rule-based systems to neural semantic parsers and, more recently, LLM-based agents, with each stage improving flexibility while introducing new limitations.

Rule-based methods. Early approaches explicitly mapped natural language to database queries using handcrafted rules, syntactic templates, and database-specific knowledge Zhong et al. (2017); Zelle & Mooney (1996); Li & Jagadish (2014). Although interpretable and controllable, they scaled poorly across domains and struggled with linguistic variation, ambiguous intents, and complex schemas.

Neural network-based methods. Deep neural models shifted Text-to-SQL toward learned semantic parsing Cao et al. (2021); Cai et al. (2021); Hui et al. (2022). By learning representations of questions and schemas, attention and schema-aware encoding improved query–database modeling. However,

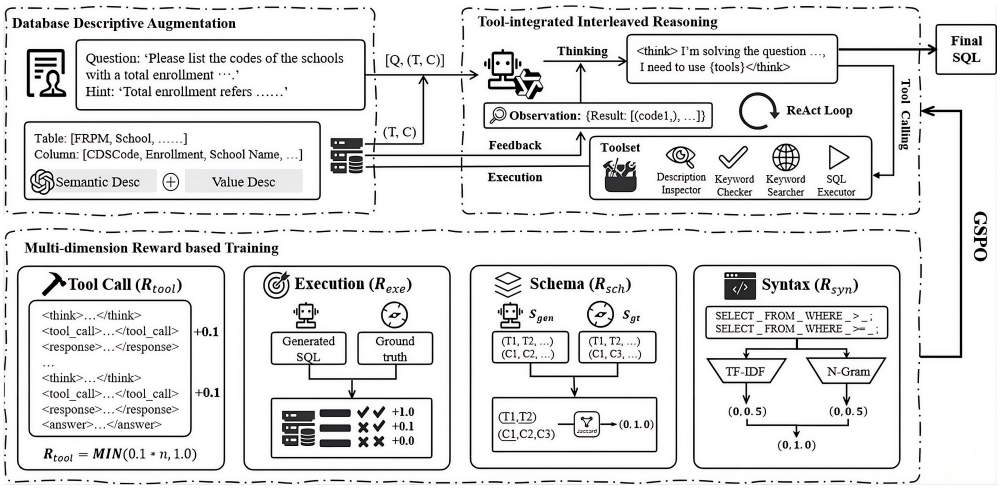

Figure 2: The overall architecture of STIR-SQL, including Database descriptive augmentation, Tool-integrated interleaved reasoning and Multi-dimension reward based training.

reliance on task-specific architectures Guo et al. (2019); Wang et al. (2019) and extensive high-quality supervision Choi et al. (2021) limited scalability and cross-domain generalization.

LLM-based methods. General-purpose LLMs Devlin et al. (2019); Raffel et al. (2020); Achiam et al. (2023) redirected research from specialized architectures toward pretrained reasoning and generation. Early work explored few-shot prompting Brown et al. (2020), least-to-most prompting Kojima et al. (2022), and Chain-of-Thought decomposition Wei et al. (2022), demonstrating complex SQL generation with limited task-specific training. However, direct prompting becomes unreliable over large schemas because of weak schema grounding and long-context reasoning difficulties. Multi-stage pipelines therefore separate schema linking, SQL generation, and post-processing Dong et al. (2023); Pourreza & Rafiei (2023); Sun et al. (2023); Lee et al. (2024); Pourreza & Rafiei (2024), achieving strong accuracy at the cost of substantial inference overhead and error propagation from early decisions.

RL-based methods. Methods based on PPO Schulman et al. (2017) and GRPO Guo et al. (2025) optimize SQL generation using execution feedback, structural similarity, and task-specific rewards Ma et al. (2025); Pourreza et al. (2025); Yao et al. (2025). Reinforcement learning directly targets task performance and offers greater optimization flexibility than supervised fine-tuning. Nevertheless, most methods emphasize the final SQL while retaining predefined pipelines or static intermediate representations.

Tool-integrated reasoning methods. Tool-integrated reasoning enables models to actively explore databases and incorporate environmental observations during generation. SKYRL-SQL Liu et al. first introduced reinforcement-learned multi-turn tool interaction for iterative probing, refinement, and verification. ReEx-SQL Dai et al. (2026) adds execution-aware feedback during decoding, while MTIR-SQL Xu et al. (2025) further optimizes interactive trajectories with stepwise execution feedback.

Despite these advances, multi-stage, RL-based, and TIR methods do not fully unify schema linking, SQL generation, post-processing, and execution verification. Multi-stage methods propagate static linking errors, RL methods mainly optimize final outputs, and TIR methods often leave grounding partially separated. STIR-SQL addresses this gap through a single adaptive trajectory integrating schema grounding, tool interaction, query generation, and self-correction for dynamic database interaction and more effective error recovery.

3 METHODOLOGY

3.1 OVERVIEW

As shown in Fig. 2, STIR-SQL takes a natural language question Q and primary database schemas S (comprising tables T and columns C) as input, and outputs a final executable SQL query with its interleaved reasoning trajectory. This is achieved through three core components:

First, STIR-SQL performs a Database descriptive augmentation method. This method enriches the raw schemas with detailed descriptive information, constructing an augmented interaction environment that provides high-quality semantic grounding for tool calling. Second, within this augmented environment, the model performs Tool-integrated interleaved reasoning. It dynamically resolves semantic ambiguities through the synergy of the model's internal reasoning and its active interaction with the database via specialized tools. Third, to optimize this capability, we employ Multi-dimension reward based training to train the model. The training process is implemented through GSPO algorithm Zheng et al. (2025), where the reward is derived from evaluating the entire reasoning trajectory across multiple aspects, including tool compliance, schema accuracy, and syntactic structure.

3.2 Database descriptive augmentation

Traditional database augmentation methods Pourreza & Rafiei (2023); Talaei et al. (2024) in Text-to-SQL typically vectorize the entire database to provide grounding support for table and column selection in schema link. However, this approach incurs indexing overhead and retrieval latency. To address these issues, we propose a lightweight Database descriptive augmentation method that generates concise descriptive statements for every column in the databases.

By converting the meanings of metadata into natural language, we establish a precise grounding context for the model. This method offers two primary advantages: first, these descriptive statements serve as real-time observations within the Tool-integrated interleaved reasoning process, enabling the model to dynamically understand the database schemas and mitigate semantic ambiguity; second, the approach possesses superior scalability for large-scale databases. Unlike conventional methods Talaei et al. (2024) that require indexing and vector embeddings for millions of specific database values, our approach processes only the tables and columns, thereby reducing system complexity and eliminating the high-latency bottlenecks associated with searching through massive database contents.

Specifically, we generate descriptive statements for each column encompassing the following two types of information:

- **Semantic Descriptions**: The functional meaning of a column generated by an LLM, which clarifies the intent of the data (e.g., 'CDSCode is the unique id for each school').
- **Value Descriptions**: Technical metadata obtained through fixed SQL query templates, including specific value formats (e.g., YYYY-MM-DD) and the presence of NULL values.

By integrating these descriptive statements, the model can actively interact with the augmented environment to ground its reasoning steps in verified data facts.

3.3 Tool-integrated interleaved reasoning

Most of models follow a typical logic when performing tool calling: they analyze the user's intent, execute one or more pre-determined tools, and provide the final answer based on the execution results. However, in complex Text-to-SQL tasks, this typical approach often leads to unnecessary tool calls and repeated failures, as it lacks the flexibility to handle error recovery or iterative evidence gathering.

To address these issues, we introduce the Tool-integrated interleaved reasoning capability of STIR-SQL. This capability empowers the model with a ReAct-style "Think-Act-Observe" loop Yao et al. (2022), conceptualizing the problem-solving process as a series of reasoning-driven interactions within the database environment. Each turn t is formally decomposed into three core components:

Reasoning (r_t): An internal thought step where the model analyzes the current problem state, reflects on prior observations, and strategies the next action.

Action (a_t): An operation selected from a predefined action space $\mathcal{A}$, including providing the final answer or calling one of the following four tools designed for SQL tasks:

- **Table/column description inspection tool**: It retrieves specific semantic and value descriptions by mapping the abstract schemas to descriptions generated by an LLM. This helps the model perform direct semantic alignment between user intent and database when encountering ambiguous columns.
- **Keyword existence check tool**: A validation probe that confirms the existence of specific keywords in designated columns. By verifying the exact data format (e.g., case sensitivity) before SQL construction, it prevents generating queries that return null results due to string mismatches in WHERE clauses.

- **Global keyword search tool**: This tool scans all columns to locate a keyword and returns its exact table and column location. It allows the model to accurately identify required join paths or filtering attributes based on data evidence rather than semantic inference alone.
- **SQL execution tool**: It leverages the database engine as an external verifier to execute candidate queries. By observing actual outputs or error messages, the model can perform self-correction and resolve edge cases that are beyond the scope of static analysis.

Observation (o_t): After an action is executed, the environment returns corresponding feedback. For a tool call, this is the tool's execution result (e.g., a query result set); for the answer action, it signifies the termination of the trajectory.

Unlike the typical logic, our model performs reasoning (r_t) before an action (a_t), and upon receiving the observation (o_t), it engages in further reasoning (r_{t+1}) rather than immediately finalizing the answer. This allows the model to dynamically decide whether to call another tool or give the final answer to conclude the process. Through this iterative loop, the model constructs a complete interaction trajectory τ:

$$\tau = (r_1, a_1, o_1, r_2, a_2, o_2, \ldots, r_T, a_T, o_T) \tag{1}$$

In STIR-SQL, this process is formalized as follows:

$$(r_{t+1}, a_t) \sim \pi_\theta(\cdot \mid r_t, o_{t-1}), \quad o_t = T(a_t), \quad a_t \in \{\text{tool}, \text{answer}\}. \tag{2}$$

where r_t represents the intermediate reasoning state, a_t denotes the action, and o_t is the observation from tool execution T (or termination for the answer action). The objective is to learn an optimal policy $\pi_\theta(a_t \mid Q, S, \tau_{<t})$ that maximizes the expected cumulative reward, ensuring the model maintains a deep understanding of the problem-solving context before generating the final SQL query.

3.4 Multi-dimension reward based training

Traditional reward for Text-to-SQL based on simple binary reward: a reward of $+1$ when the SQL execution result is correct and 0 when error. This reward design is is suitable for typical "end-to-end" reasoning process, but is not enough for Tool-integrated interleaved reasoning. For a better guide to the training of the model with long interleaved reasoning trajectory, we introduce Multi-dimension reward based training of STIR-SQL through two primary components: Multi-dimension reward design and Multi-dimension reward based reinforcement learning.

3.4.1 Multi-dimension reward design

To accommodate multi-tool calling scenario and the requirement for complex SQL generation in the Text-to-SQL domain, we design a multi-dimension reward to guide the model training. The total reward is a weighted sum of the following four components:

Tool call reward (R_{tool}): To encourage the model to utilize the provided tools effectively, a partial reward is granted for each successful tool calling. A successful call is defined as one that follows the correct format and returns valid observations from the environment, while failed or malformed calls receive no score. This component is capped at a maximum value of 1.0.

Execution accuracy reward (R_{exe}): This reward evaluates the final generated SQL query Y by comparing its execution results with those of the ground truth SQL Y_{gt}. Let $\text{Exe}(\cdot)$ denote the execution function on the database. The scoring logic is defined as follows:

$$R_{exe} = \begin{cases} 1.0 & \text{if Exe}(Y) = \text{Exe}(Y_{gt}) \\ 0.1 & \text{if Exe}(Y) \neq \text{Exe}(Y_{gt}) \\ 0 & \text{if Exe}(Y) \text{ Error} \end{cases} \tag{3}$$

Schema accuracy reward (R_{sch}): This reward measures the model's ability to identify the correct database structures. We calculate the Jaccard similarity between the schemas utilized in the generated SQL (S_{gen}) and ground truth SQL (S_{gt}):

$$R_{sch} = \frac{|S_{gen} \cap S_{gt}|}{|S_{gen} \cup S_{gt}|} \tag{4}$$

The score reaches 1.0 when the selected schemas perfectly align with the ground truth.

Syntax similarity reward (R_{syn}): To encourage the generation of syntactically sound and logically structured SQL, we evaluate the similarity between the skeletons of the generated and ground truth queries. A SQL skeleton is an abstraction where specific values and identifiers are replaced by placeholders (e.g., SELECT _ FROM _ WHERE _ ¿ _). The reward is calculated as the average of TF-IDF Robertson & Jones (1976) cosine similarity and N-gram Jelinek (1980) Jaccard similarity:

$$R_{\text{syn}} = 0.5\,\text{Sim}_{\text{TF-IDF}}(K_{\text{gen}}, K_{\text{gt}}) + 0.5\,\text{Sim}_{\text{N-gram}}(K_{\text{gen}}, K_{\text{gt}}). \tag{5}$$

where K_{gen} and K_{gt} denote the skeletons of the generated and ground truth SQL, respectively. Specifically, $\text{Sim}_{\text{TF-IDF}}$ measures the similarity of the global keyword distribution using vector representations, while $\text{Sim}_{\text{N-gram}}$ assesses the local structural order by comparing sets of 3-gram token sequences. This dual-metric approach ensures that the model captures both the presence of essential SQL keywords and the correct syntactic sequence.

3.4.2 Multi-dimension reward based reinforcement learning

Due to the efficiency and effectiveness of Group sequence policy optimization (GSPO) Zheng et al. (2025) in structured reasoning tasks, particularly its ability to stabilize training for MoE architectures Zhou et al. (2022), we adopt it as the core reinforcement learning algorithm for STIR-SQL. Unlike traditional methods that operate at the token level, GSPO Zheng et al. (2025) introduces sequence-level importance ratios to address training instability and improve optimization efficiency. For each input x (comprising the natural language question Q and database schemas S), the model generates a group of G candidate sequences $\{y_1, y_2, \ldots, y_G\}$. The sequence-level importance ratio $s_i(\theta)$ is defined as:

$$s_i(\theta) = \exp\left(\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\log\frac{\pi_\theta(y_{i,t} \mid x, y_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t} \mid x, y_{i,<t})}\right). \tag{6}$$

The optimization objective of GSPO Zheng et al. (2025) is formulated as:

$$\mathcal{J}_{\text{GSPO}}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\min\Big(s_i(\theta)\hat{A}_i, \text{clip}(s_i(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_i\Big)\right]. \tag{7}$$

where the expectation $\mathbb{E}$ is taken over the input $x \sim \mathcal{D}$ and the corresponding group of G rollouts generated by the old policy $\pi_{\theta_{\text{old}}}$, $s_i(\theta)$ represents the sequence-level importance ratio which acts as the geometric mean of token-level ratios for a sequence y_i of length $|y_i|$, $\hat{A}_i$ denotes the group-relative advantage computed by normalizing the rewards obtained by the i-th sequence within its group to stabilize the policy update, x symbolizes the input context sampled from the training distribution $\mathcal{D}$, and ϵ is a hyperparameter for the clipping mechanism that constrains the update step to ensure a stable training trajectory.

4 Experiment

4.1 Experiment setup

4.1.1 Datasets and metrics

We evaluate STIR-SQL on the in-domain BIRD benchmark Li et al. (2023) and the out-of-domain Spider Yu et al. (2018) and Spider-DK Gan et al. (2021) benchmarks. Performance is measured by Execution Accuracy (EX), which compares the execution results of predicted and reference SQL queries.

4.1.2 Baselines

We compare against four categories: general LLMs, including Codestral Mistral (2024) and GPT-4o Achiam et al. (2023); task-specific SFT/RL models, including SQL-R1 Yao et al. (2025), Omni-SQL Li et al. (2025), Think2SQL Papicchio et al. (2025), DeepRetrieval Jiang et al. (2025), Reasoning-SQL Pourreza et al. (2025), CodeS Li et al. (2024), and XiYan-SQL Liu et al. (2025b); TIR methods, including SKYRL-SQL Liu et al., ReEx-SQL Dai et al. (2026), and MTIR-SQL Xu et al. (2025); and multi-stage pipelines, including Distillery + GPT-4o Maamari et al. (2024), DAIL-SQL + GPT-4 Guo et al. (2021), and CHESS + DeepSeek-Coder-33B Talaei et al. (2024).

Model	Params	OSS	Post-tr.	Trainsets	Reason	BIRD-dev	Spider-test	Spider-DK
Model Size $<$ 10B								
SQL-R1-7B Ma et al. (2025)	7B	✗	SFT+RL	Spider+BIRD	Yes	66.6	88.1	78.9
OmniSQL-7B Li et al. (2025)	7B	✓	SFT	Spider+BIRD	No	63.9	87.9	76.1
Think2SQL-7B Papicchio et al. (2025)	7B	✗	SFT+RL	BIRD	No	56.1	–	–
DeepRetrieval Jiang et al. (2025)	7B	✓	SFT+RL	Spider+BIRD	No	56.0	76.1	–
SKYRL-SQL Liu et al.	7B	✓	RL	SPIDER	Yes	–	85.2	–
ReEx-SQL Dai et al. (2026)	7B	✓	RL	BIRD	Yes	64.9	86.6	**80.2**
MTIR-SQL Xu et al. (2025)	4B	✓	RL	SPIDER+BIRD	Yes	64.4	83.4	–
10B $\leq$ Model Size $\leq$ 30B								
OmniSQL-14B Li et al. (2025)	14B	✓	SFT	Spider+BIRD	No	64.2	<u>88.3</u>	72.9
Reasoning-SQL-14B Pourreza et al. (2025)	14B	✗	SFT+RL	Spider+BIRD	Yes	66.2	81.4	73.03
CodeS Li et al. (2024)	15B	✗	SFT	BIRD	Yes	57.9	79.4	72.0
Codestral-22B Mistral (2024)	22B	✓	–	–	No	52.7	78.6	70.0
Model Size $>$ 30B or Unknown								
GPT-4o Achiam et al. (2023)	Unknown	✗	–	–	No	61.9	83.2	72.9
XiYan-SQL-32B Liu et al. (2025b)	32B	✓	SFT	Spider+BIRD	No	67.0	–	–
OmniSQL-32B Li et al. (2025)	7B	✓	SFT	Spider+BIRD	No	64.5	87.6	76.1
ExDPO-32B Zhai et al. (2025)	32B	✓	SFT+RL	Spider+BIRD	Yes	68.3	85.1	–
Distillery + GPT-4o Maamari et al. (2024)	Unknown	✗	–	–	No	67.2	–	–
DAIL-SQL + GPT-4 Gao et al. (2023)	Unknown	✗	–	–	No	54.8	86.2	–
CHESS+DS-Coder-33B Talaei et al. (2024)	33B	✗	SFT	Spider+BIRD	Yes	65.0	87.2	–
STIR-SQL (Ours)	**3B**	✓	RL	BIRD	**Yes**	**69.7**	**88.7**	<u>79.1</u>

Table 1: Comparison of execution accuracy (EX, %) on BIRD, Spider, and Spider-DK benchmarks. We report inference parameters, open-source status (OSS), post-training methods, and reasoning capabilities.

4.1.3 Implementation details

We train Qwen3 Yang et al. (2025) using GSPO Zheng et al. (2025) in Verl Sheng et al. (2024), with a batch size of 64, a learning rate of 10^{-6}, 16 rollouts per input, and at most $N = 30$ interactions. SQLite provides execution feedback, and training uses eight NVIDIA A100 GPUs.

4.2 Comparisons with baselines

Table 1 compares execution accuracy across three benchmarks. With only 3B active parameters, STIR-SQL yields three observations.

First, STIR-SQL is parameter efficient. On BIRD-dev Li et al. (2023), it achieves **69.7%** EX, exceeding ExDPO-32B Zhai et al. (2025) (68.3%) and XiYan-SQL-32B Liu et al. (2025b) (67.0%). Its single trajectory needs one LLM call per query, avoiding repeated multi-stage calls and improving inference efficiency.

Second, pure reinforcement learning is effective. On BIRD-dev Li et al. (2023), STIR-SQL surpasses CHESS + DeepSeek-Coder-33B Talaei et al. (2024) (65.0%), Distillery + GPT-4o Maamari et al. (2024) (67.2%), ReEx-SQL Dai et al. (2026) (64.9%), and MTIR-SQL Xu et al. (2025) (64.4%) with a smaller inference footprint. Rather than scale or extra interactions, the gains stem from single-trajectory reasoning and multi-dimensional rewards that jointly optimize schema understanding, SQL generation, and execution verification.

Third, STIR-SQL generalizes across domains, achieving **88.7%** on Spider-test Yu et al. (2018) and **79.1%** on Spider-DK Gan et al. (2021). Its BIRD Li et al. (2023) and Spider Yu et al. (2018) results show adaptation to diverse schemas and query complexities through a ReAct Yao et al. (2022) trajectory that interleaves reasoning, tool calls, and observations without domain-specific rules.

4.3 Ablation study

Table 2 evaluates the contribution of each STIR-SQL component.

4.3.1 w/o Database augmentation

Replacing augmented descriptions with raw schemas lowers performance by 0.6%. Cryptic or abbreviated column names force the model to infer meanings through exploration and can confuse similarly named columns with different functions, leading to incorrect SQL.

Variant	Simple	Moderate	Challenging	BIRD-dev	Δ
STIR-SQL (Full)	**74.5**	**64.0**	**57.2**	**69.7**	–
w/o Database augmentation	73.6	64.0	56.6	69.3	-0.6
w/o GSPO	73.0	62.7	55.9	68.3	-1.4
w/o Interleaved reasoning	71.5	60.6	51.0	66.2	-3.5
w/ Single SQL execution tool	73.2	62.1	53.8	68.0	-1.7
w/ Simple reward	72.8	61.9	53.1	67.6	-2.1

Table 2: Ablation study of STIR-SQL on the BIRD benchmark. All results are percentages; Δ is the overall performance decrease relative to the full model.

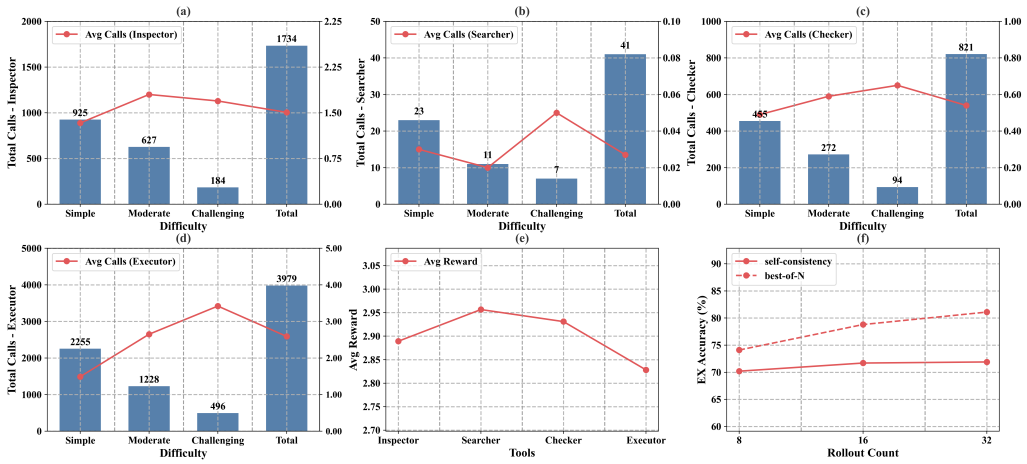

Figure 3: Experimental results overview: (a-d) calling counts of specialized tools across difficulties; (e) average rewards per tool; (f) performance scaling under varying rollout counts (N).

4.3.2 w/o GSPO

Replacing GSPO Zheng et al. (2025) with Group Relative Policy Optimization (GRPO) Guo et al. (2025) and routing replay lowers performance by 1.4%, reflecting expert-activation volatility in sparse MoE models Zhou et al. (2022). In Qwen3 Yang et al. (2025), token-level importance ratios can fluctuate between π_θ and $\pi_{\theta_{\text{old}}}$ despite routing replay. GSPO's sequence-level ratios smooth this variance and stabilize optimization.

4.3.3 w/o Interleaved reasoning

Replacing interleaved reasoning and multi-turn tool calls with direct SQL generation from the question and schemas lowers performance by 3.5%, the largest ablation drop. The ReAct Yao et al. (2022) "Think-Act-Observe" loop resolves uncertainty through interaction and revision; without it, the model cannot retrieve column descriptions or check keyword existence, increasing schema-linking and semantic errors.

4.3.4 w/ Single SQL execution tool

Keeping only SQL execution and removing description inspection, keyword checks, and global search lowers performance by 1.7%. Execution feedback remains available, but identifying database structures through trial-and-error SQL is inefficient and unreliable with ambiguous schemas or descriptions.

4.3.5 w/ Simple reward

Replacing the composite reward with $+1$ for a ground-truth-matching SQL result and 0 otherwise lowers performance by 2.1%. This sparse signal poorly guides tool use and intermediate reasoning; without schema and syntax rewards, the model struggles with table selection and SQL construction.

4.4 Tool calling analysis

We analyze tool-calling behavior to understand how interleaved reasoning supports SQL generation.

Type	Med	Mode	Max	Q1	Q3	Avg
Correct answer	3.0	3.0	12.0	3.0	4.0	4.01
Incorrect answer	4.0	3.0	18.0	3.0	7.0	5.21

Table 3: Statistics of tool calling counts categorized by problem correctness on the BIRD-dev set. The data includes median (Med), mode, maximum (Max), lower quartile (Q1), upper quartile (Q3), and average (Avg).

4.4.1 Efficiency and correctness

Table 3 shows that correct trajectories use fewer tools than incorrect ones, averaging 4.01 versus 5.21 calls per problem. Successful trajectories identify relevant schemas, column relationships, and constraints with fewer redundant interactions, resulting in shorter and more efficient reasoning paths.

This finding is consistent with the STIR paradigm, which integrates tool interaction into reasoning rather than treating it as a separate stage. When the initial reasoning captures the key information, subsequent calls mainly verify targeted hypotheses. In contrast, incorrect trajectories often invoke more tools because of unresolved semantic ambiguity, inaccurate schema understanding, or invalid SQL construction. Additional feedback cannot guarantee correction once the reasoning trajectory deviates substantially, highlighting the importance of accurate initial reasoning and effective observation use.

4.4.2 Distribution across difficulty levels

Figure 3 (a–e) shows that STIR-SQL adaptively selects tools according to query requirements. The SQL execution tool is used most frequently, averaging 2.59 calls with a reward of 2.83, because it directly verifies intermediate queries and exposes errors. Its relatively low reward reflects frequent trial-and-error executions.

The table/column description inspection tool averages 1.13 calls and a reward of 2.89, supporting schema linking and semantic grounding. The keyword existence check tool averages 0.54 calls with a reward of 2.93, helping verify candidate values. Although the global keyword search tool is rarely used (0.03 calls), it achieves the highest reward of 2.96, indicating selective use for difficult schema or value ambiguities.

Total tool calls increase from simple to challenging queries, demonstrating that STIR-SQL dynamically adjusts its reasoning depth and interaction frequency rather than applying a fixed strategy.

4.5 Scalability analysis

We evaluate test-time scaling by increasing the number of candidate rollouts N. As shown in Figure 3 (f), Best-of-N accuracy increases nearly monotonically, indicating that diverse sampled trajectories provide complementary candidates and improve the probability of finding a correct SQL query.

Self-consistency initially improves by 1.5% but gradually saturates, gaining only 0.2% at the largest rollout counts. This suggests that additional samples may continue to discover correct solutions, while majority voting provides limited benefit after a dominant reasoning pattern emerges.

Overall, increasing rollouts improves performance, but the gain depends on the selection strategy. A moderate rollout count offers a practical balance between accuracy and computational cost, whereas excessive scaling yields diminishing returns.

The remaining supplementary experiments are shown in the Appendix.

5 Conclusion

We introduce STIR-SQL, a Text-to-SQL agent trained through pure reinforcement learning to perform single-trajectory interleaved reasoning. GSPO and a multi-dimensional reward teach it to use four tools for schema understanding and iterative SQL correction. On BIRD, Spider, and Spider-DK, STIR-SQL achieves competitive execution accuracy, strong generalization, and low generation latency with only 3B active parameters, outperforming many larger models and complex multi-stage systems. These findings highlight the potential of pure reinforcement learning and internal reasoning to narrow the gap to human-expert SQL generation for practical structured database retrieval.

AI USE STATEMENT

Generative AI tools were used solely for language polishing and grammar checking of the manuscript. The core research ideas, methodological design, and all experiments were developed and conducted by the authors. The authors reviewed all AI-assisted language edits and take responsibility for the final manuscript.

REPRODUCIBILITY STATEMENT

An anonymized copy of the project code is submitted with the supplementary materials. The main paper specifies the datasets, evaluation metrics, model, training procedure, and key hyperparameters. Appendix A reports reward-component training dynamics and reward ablations; Appendix B analyzes tool use, self-correction, and alternative reasoning frameworks; Appendix C provides inference-efficiency results and a case study; and Appendix D gives the prompt template used for training and evaluation. These materials and the code document the settings needed to reproduce the reported experiments.

REFERENCES

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

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

Ruichu Cai, Jinjie Yuan, Boyan Xu, and Zhifeng Hao. Sadga: Structure-aware dual graph aggregation network for text-to-sql. *Advances in Neural Information Processing Systems*, 34:7664–7676, 2021.

Ruisheng Cao, Lu Chen, Zhi Chen, Yanbin Zhao, Su Zhu, and Kai Yu. Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations. *arXiv preprint arXiv:2106.01093*, 2021.

DongHyun Choi, Myeong Cheol Shin, EungGyun Kim, and Dong Ryeol Shin. Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases. *Computational Linguistics*, 47(2):309–332, 2021.

Yaxun Dai, Wenxuan Xie, Xialie Zhuang, Tianyu Yang, Ziyi Liu, Haiqin Yang, Yiying Yang, Yuhang Zhao, Pingfu Chao, and Wenhao Jiang. Reex-sql: Reasoning with execution-aware reinforcement learning for text-to-sql. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 824–847, 2026.

Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-sql: a survey of what we have and what we expect. *arXiv preprint arXiv:2208.10099*, 2022.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers)*, pp. 4171–4186, 2019.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*, 2023.

Yujian Gan, Xinyun Chen, and Matthew Purver. Exploring underexplored limitations of cross-domain text-to-sql generalization. *arXiv preprint arXiv:2109.05157*, 2021.

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

Aibo Guo, Xiang Zhao, and Wubin Ma. Er-sql: Learning enhanced representation for text-to-sql using table contents. *Neurocomputing*, 465:359–370, 2021.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025.

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

Binyuan Hui, Ruiying Geng, Lihan Wang, Bowen Qin, Bowen Li, Jian Sun, and Yongbin Li. S2-sql: Injecting syntax to question-schema interaction graph encoder for text-to-sql parsers. *arXiv preprint arXiv:2203.06958*, 2022.

Frederick Jelinek. Interpolated estimation of markov source parameters from sparse data. In *Proc. Workshop on Pattern Recognition in Practice, 1980*, 1980.

Pengcheng Jiang, Jiacheng Lin, Lang Cao, Runchu Tian, SeongKu Kang, Zifeng Wang, Jimeng Sun, and Jiawei Han. Deepretrieval: Hacking real search engines and retrievers with large language models via reinforcement learning. *arXiv preprint arXiv:2503.00223*, 2025.

George Katsogiannis-Meimarakis and Georgia Koutrika. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936, 2023.

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

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*, 2024.

Fei Li and Hosagrahar V Jagadish. Nalir: an interactive natural language interface for querying relational databases. In *Proceedings of the 2014 ACM SIGMOD international conference on Management of data*, pp. 709–712, 2014.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*, 2025.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357, 2023.

Shu Liu, Alan Zhu, Sumanth Hegde, Shiyi Cao, Shuo Yuan, Samion Suwito, Tyler Griggs, Matei Zaharia, Joseph E Gonzalez, and Ion Stoica. Skyrl-sql: Multi-turn sql data agents via rl. In *First Workshop on Multi-Turn Interactions in Large Language Models*.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of text-to-sql in the era of llms: Where are we, and where are we going? *IEEE Transactions on Knowledge and Data Engineering*, 2025a.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, et al. Xiyan-sql: A novel multi-generator framework for text-to-sql. *arXiv preprint arXiv:2507.04701*, 2025b.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. Sql-r1: Training natural language to sql reasoning model by reinforcement learning. *arXiv preprint arXiv:2504.08600*, 2025.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. *arXiv preprint arXiv:2408.07702*, 2024.

AI Mistral. Codestral: Hello, world. *Section: news. Brundage, Mira Murati, Katie Mayer, Peter Welinder, Bob McGrew, Dario Amodei, Sam McCandlish, Ilya Sutskever, and Wojciech Zaremba. 2021a. Evaluating large language models trained on code. Preprint*, 2024.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2sql: Reinforce llm reasoning capabilities for text2sql. *arXiv preprint arXiv:2504.15077*, 2025.

Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348, 2023.

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

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, Sercan Arik, et al. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *arXiv preprint arXiv:2503.23157*, 2025.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*, 2022.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of machine learning research*, 21(140):1–67, 2020.

Stephen E Robertson and K Sparck Jones. Relevance weighting of search terms. *Journal of the American Society for Information science*, 27(3):129–146, 1976.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Zhihao Shen, Kang Yang, Xi Zhao, Jianhua Zou, and Wan Du. Deepapp: A deep reinforcement learning framework for mobile application usage prediction. *IEEE Transactions on Mobile Computing*, 22(2):824–840, 2021.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Computing Surveys*, 58(2):1–37, 2025.

Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, et al. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *arXiv preprint arXiv:2306.00739*, 2023.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*, 2024.

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

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

Zekun Xu, Siyu Xia, Chuhuai Yue, Jiajun Chai, Mingxue Tian, Xiaohan Wang, Wei Lin, Haoxuan Li, and Guojun Yin. Mtir-sql: Multi-turn tool-integrated reasoning reinforcement learning for text-to-sql. *arXiv preprint arXiv:2510.25510*, 2025.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *arXiv preprint arXiv:2505.20315*, 2025.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

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

Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. Excot: Optimizing reasoning for text-to-sql with execution feedback. *arXiv preprint arXiv:2503.19988*, 2025.

Chujie Zheng, Shixuan Liu, Mingze Li, Xiong-Hui Chen, Bowen Yu, Chang Gao, Kai Dang, Yuqiong Liu, Rui Men, An Yang, et al. Group sequence policy optimization. *arXiv preprint arXiv:2507.18071*, 2025.

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

Yanqi Zhou, Tao Lei, Hanxiao Liu, Nan Du, Yanping Huang, Vincent Zhao, Andrew M Dai, Quoc V Le, James Laudon, et al. Mixture-of-experts with expert choice routing. *Advances in Neural Information Processing Systems*, 35:7103–7114, 2022.