Graph-Reward-SQL: Execution-Free Reinforcement Learning for Text-to-SQL via Graph Matching and Stepwise Reward

Han Weng[1,2]*, Puzhen Wu[2], Longjie Cui[2]*, Yi Zhan[2]*, Boyi Liu[2], Yuanfeng Song[2], Dun Zeng[2], Yingxiang Yang[2], Qianru Zhang[2], Dong Huang[2,3]†, Xiaoming Yin[2], Yang Sun[2]†, Xing Chen[2]

[1] Beijing University of Posts and Telecommunications, China

[2] ByteDance, China

[3] Institute of Data Science, National University of Singapore

Abstract

Reinforcement learning (RL) has been widely adopted to enhance the performance of large language models (LLMs) on Text-to-SQL tasks. However, existing methods often rely on execution-based or LLM-based Bradley–Terry reward models. The former suffers from high execution latency caused by repeated database calls, whereas the latter imposes substantial GPU memory overhead, both of which significantly hinder the efficiency and scalability of RL pipelines. To this end, we propose a novel reward model framework for RL-based Text-to-SQL named Graph-Reward-SQL, which employs the GMNScore outcome reward model. We leverage SQL graph representations to provide accurate reward signals while significantly reducing time cost and GPU memory usage. Building on this foundation, we further introduce StepRTM, a stepwise reward model that provides intermediate supervision over Common Table Expression (CTE) subqueries. This encourages both functional correctness and readability of SQL. Extensive comparative and ablation experiments on standard benchmarks, including Spider and BIRD, demonstrate that our method consistently outperforms existing reward models.

1 Introduction

Text-to-SQL (Tai et al., 2023; Li et al., 2024b; Shi et al., 2025) aims to translate natural language into structured database queries and plays a crucial role in democratizing data access by enabling non-technical users to interact with relational databases more effectively. A significant body of work has focused on fine-tuning foundational models, with recent studies showing that Reinforcement Learning (RL) can effectively enhance model performance (Pourreza et al., 2025b; Berdnyk and Collery, 2025; Ma et al., 2025). Among these efforts, the careful design of the reward model is a crucial challenge, as the quality of the reward signal directly influences policy optimization during fine-tuning.

In RL-based Text-to-SQL approaches, execution accuracy remains a dominant signal (Nguyen et al., 2025; Ma et al., 2025; Pourreza et al., 2025b; Berdnyk and Collery, 2025), providing intuitive feedback based on query correctness. Additionally, the Bradley–Terry reward model (BTRM) (Christiano et al., 2017) has been adapted for code generation by deriving preference pairs from execution outcomes (Zeng et al., 2025a). Structural rewards based on abstract syntax tree (AST) have also been explored to capture syntactic similarity (Shojaee et al., 2023). However, each approach has significant limitations in the Text-to-SQL tasks. Execution-based rewards introduce significant latency due to runtime database access. The LLM-based BTRM incurs high computational and GPU memory costs, which limits its scalability. AST matching-based similarity is prone to false negatives, where syntactically divergent queries that are semantically equivalent are penalized, leading to inaccurate reward signals. These limitations point to a key challenge in RL-based Text-to-SQL: designing an efficient reward model that can replace execution-based signals without compromising performance.

To address the above limitations, we introduce Graph-Reward-SQL, a novel reward model framework for RL-based Text-to-SQL. This framework incorporates two complementary reward models: Graph Matching Network Score (GMNScore) and Stepwise Relational Operator Tree Match (StepRTM). GMNScore serves as an outcome reward model, which evaluates the generated SQL queries using the Graph Matching Network (GMN) without requiring execution. GMN utilizes learned graph embeddings to assess functional equivalence, capturing the deep semantics of SQL queries (Zhan et al., 2025). In contrast to execution-based re-

*Work was done during the internship at ByteDance.

†Corresponding authors

Findings of the Association for Computational Linguistics: EMNLP 2025, pages 12917–12943
November 4-9, 2025

wards, GMNScore eliminates the need for costly database executions, resulting in a significant speed-up. Furthermore, compared to LLM-based Bradley-Terry reward models (BTRM), GMNScore substantially reduces GPU memory consumption due to the lightweight architecture of GMN. Additionally, StepRTM provides intermediate feedback through a stepwise reward mechanism that evaluates the generation of Common Table Expression (CTE) subqueries, complementing GMNScore.

The above design offers three notable advantages. (i) Superior Training Efficiency: Our method significantly reduces time cost and GPU memory usage compared to existing outcome reward models, leading to enhanced overall training efficiency for RL. (ii) Intermediate Feedback Integration: Unlike existing reward models that focus solely on outcome result, our framework incorporates intermediate evaluation by leveraging the structure of CTE SQL. This provides richer feedback during training, improving performance and readability. (iii) Strong Empirical Performance: Extensive ablation studies and evaluations on the Spider (Yu et al., 2018) and BIRD (Li et al., 2024b) Text-to-SQL benchmarks validate the superiority of our reward models. The results consistently demonstrate that our approach outperforms multiple strong reward model baselines, highlighting its effectiveness.

Our contributions are summarized as follows:

- We propose GMNScore, an outcome reward model that leverages GMN to replace execution-based rewards, achieving both higher efficiency and better performance.
- We design a novel stepwise reward model StepRTM, which utilizes CTE SQL to deliver stepwise supervision by matching each subquery, resulting in improved accuracy and readability.
- Extensive experiments show that our reward models consistently improve performance while maintaining high inference efficiency and low GPU memory consumption.

2 Related Work

Text-to-SQL. Text-to-SQL is a key task in Natural Language Processing (NLP) that involves transforming queries expressed in natural language into executable SQL queries (Tai et al., 2023; Li et al., 2024b; Shi et al., 2025). With the increasing deployment of large language models (LLMs), agentic frameworks (Wang et al., 2025; Pourreza et al., 2025a; Lei et al., 2024) have been introduced to enhance Text-to-SQL tasks. These frameworks enable LLMs to interact with databases through iterative reasoning and external tools. Code Foundation Models such as DeepSeek-Coder (Guo et al., 2024) and Qwen2.5-Coder (Hui et al., 2024) provide the backbone for these agentic systems, enabling structured reasoning and code generation. Several approaches aim to improve LLM performance in Text-to-SQL tasks, including direct fine-tuning (Li et al., 2024a; Yang et al., 2024; Pourreza and Rafiei, 2024), as well as techniques such as prompt design (Pourreza and Rafiei, 2023; Dong et al., 2023; Gao et al., 2024a), self-consistency (Gao et al., 2024a) and schema linking (Guo et al., 2019; Wang et al., 2020; Lei et al., 2020; Lee et al., 2025) to further optimize results.

Reinforcement Learning and Reward Model. RL has become an important paradigm for effectively fine-tuning Code Foundation Models. Policy optimization methods, such as Proximal Policy Optimization (PPO) (Schulman et al., 2017) and Group Relative Policy Optimization (GRPO) (Shao et al., 2024), have been explored. However, the effectiveness of RL training heavily relies on the quality of reward signals, making the design of reward models a critical aspect (Trella et al., 2023). Several contributions to RL-based code generation have advanced reward model strategies. Notable works include CodeRL (Le et al., 2022), which leverages execution feedback; PPOCoder (Shojaee et al., 2023), integrating semantic matching of abstract syntax trees; and AceCoder (Zeng et al., 2025a), applying an LLM-based Bradley-Terry Reward Model.

The execution-based reward model for Text-to-SQL was initially used by (Zhong et al., 2017). Recent advancements have introduced continuous reward scores based on keyword matching (Nguyen et al., 2025) and leveraged LLMs to generate reward function candidates and iteratively refine (Berdnyk and Collery, 2025). Alongside these developments, reasoning models such as DeepSeek-R1 (Guo et al., 2025) have advanced RL in reasoning tasks, leading to the introduction of more sophisticated reward model designs. For example, SQL-R1 (Ma et al., 2025) incorporates format and length constraints, while Reasoning-SQL (Pourreza et al., 2025b) employs more complex reward structures, such as schema linking feedback, n-gram similarity scores, and LLM-based judgment. Despite these enhancements, execution-based reward continue to play a central role in the

Reward Model	Modeling Basis	Granularity	Time Cost ↓	GPU Usage↓	Perf. Rank (1=Best)
EX (Pourreza et al., 2025b)	Execution	Outcome	Slow	N/A	3
BTRM (Zeng et al., 2025a)	LLM	Outcome	Moderate	High	5
AstPM (Shojaee et al., 2023)	AST Matching	Outcome	Fast	N/A	6
RelPM (Zhan et al., 2025)	ROT Matching	Outcome	Fast	N/A	4
GMNScore (Ours)	GMN	Outcome	Fast	Low	2
+ StepRTM (Ours)	ROT Matching	Stepwise	Fast	N/A	**1**

Table 1: Comparison of Reward Models in RL for Text-to-SQL Tasks. Our proposed GMNScore and StepRTM achieve better performance while significantly reducing time and memory costs.

above-mentioned approaches.

Current methods overlook the computational overhead of execution-based and LLM-based reward models and fail to fully exploit the deep semantic structure of SQL queries. Additionally, these approaches focus solely on evaluating the final generated SQL, neglecting the potential of leveraging intermediate supervision signals throughout the SQL generation process. To address these issues, we propose an execution-free outcome reward model and a stepwise reward mechanism. These methods significantly reduce computational overhead while providing more effective reward signals for RL-based Text-to-SQL tasks.

3 Preliminaries

3.1 Problem Formulation

In the standard Text-to-SQL setting, let x denote a natural language query, $\hat{q}$ and $q^\star$ represent the generated SQL and reference SQL query, respectively. In this work, we mainly use Proximal Policy Optimization (PPO) (Schulman et al., 2017), which optimizes the policy model π_θ by maximizing:

$$\mathcal{J}(\theta) = \mathbb{E}_{(x,\hat{q})\sim\mathcal{D},\hat{q}\sim\pi_\theta(\cdot|x)}[r(\hat{q},q^\star) - \beta\mathbb{D}_{\mathrm{KL}}(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid x))],$$

where π_{ref} is the reference model, β is a PPO hyperparameter and $r(\hat{q}, q^\star)$ is a reward model. Note that our method can be easily adapted to Group Relative Policy Optimization (GRPO) (Shao et al., 2024), as detailed in the Appendix D.

3.2 Summary of Existing Reward Models

Recognizing the great importance of reward models in RL, we discuss three types of main reward models. As summarized in Table 1, we compare these models with our proposed reward models in terms of time cost and GPU memory usage during inference. Additionally, the final performance of all reward models is evaluated and ranked, as described in Section 6.1. Detailed information on these comparisons can be found in the Appendix G.

Execution Accuracy (EX). For the Text-to-SQL tasks, the execution accuracy serves as the most direct reward signal, providing a discrete score based on whether the generated SQL query yields the correct result upon execution. We use a discrete reward model with finer-grained feedback based on syntax error (Pourreza et al., 2025b) and runtime diagnostics following (Shojaee et al., 2023). Given a generated SQL $\hat{q}$ and reference SQL $q^\star$, the formulation is listed as:

$$r_{\mathrm{EX}}(\hat{q}, q^\star) = R_{exec} + R_{syntax} + R_{runtime}$$

However, the EX has notable limitations. When the database contains poor quality data (e.g., limited, missing, or inconsistent entries) or structural issues (e.g., redundancy or anomalies), different queries may produce identical results (Zhong et al., 2020). Test Suite (TS) (Zhong et al., 2020) attempted to address this issue, but as shown in (Zhan et al., 2025), false positives and false negatives remain unavoidable. Additionally, repeatedly executing SQL queries introduces significant computational overhead, increasing training time. More details about EX are provided in the Appendix F.

Bradley-Terry Reward Model (BTRM). Given a natural language input x and a candidate SQL query y, we define the reward model as $r_\psi(x, y) = h_r\big(\mathcal{M}_\theta(x, y)\big)$, with a pretrained language model $\mathcal{M}_\theta$ and a reward head h_r. The training process uses preference pairs based on execution correctness: $\mathcal{D} = \{(x_i, y_i^+, y_i^-)\}_{i=1}^N$, where y_i^+ executes correctly and y_i^- fails or returns an incorrect result (Zeng et al., 2025b). The objective is to minimize the Bradley-Terry log-likelihood (Bradley and Terry, 1952) as follows:

$$-\sum_{i=1}^{N} \log \frac{\exp\big(r_\psi(x_i, y_i^+)\big)}{\exp\big(r_\psi(x_i, y_i^+)\big) + \exp\big(r_\psi(x_i, y_i^-)\big)}$$

This model learns to assign higher scores to correct queries, providing a dense proxy reward model for

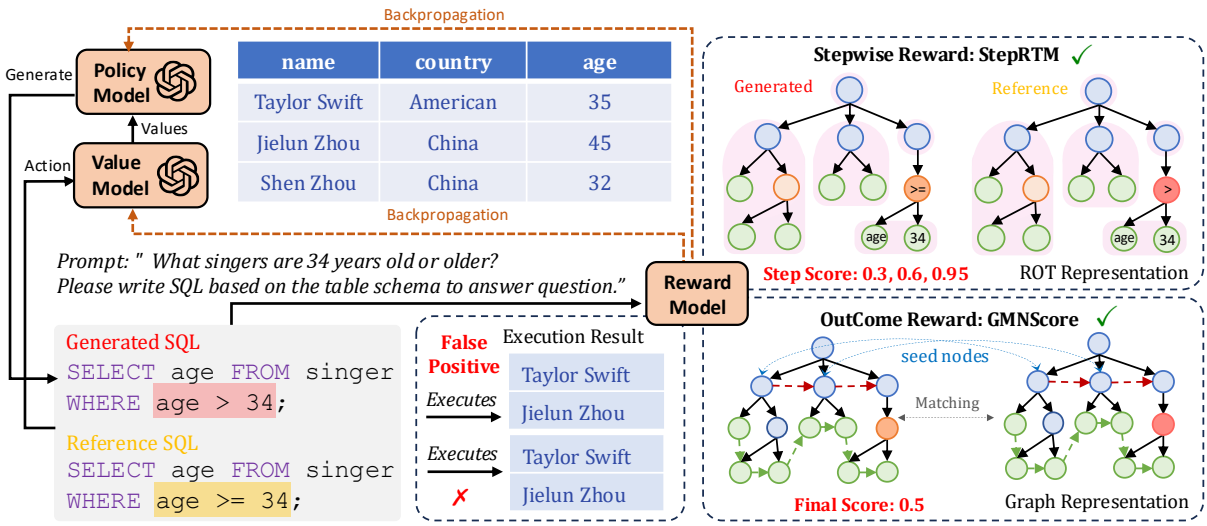

Figure 1: Graph-Reward-SQL employs PPO, where rewards drive policy updates. An example illustrates a limitation of EX: the generated SQL 'WHERE age > 34' and the reference SQL 'WHERE age >= 34' produce identical results despite their semantic difference. In contrast, our proposed GMNScore leverages graph representation to capture deep semantic similarity. Additionally, a stepwise reward model, StepRTM, that is tailored for CTE SQL addresses the lack of intermediate rewards. A mock stepwise score is provided for illustration; see Figure 2 for more details.

RL (Christiano et al., 2017). In contrast to EX, BTRM enables more efficient policy training by eliminating the need to query databases. However, the large parameter size of LLM-based BTRM significantly increases GPU memory usage.

Matching-based Reward. In (Nguyen et al., 2025), rule-based keyword matching is used for scoring SQL queries, while n-gram similarity is used in Reasoning-SQL (Pourreza et al., 2025b) to capture overlapping token sequences. Matching-based methods are fast and model-free, but may assign negative rewards to semantically equivalent SQL queries that differ in syntax, which should ideally be considered correct. In broader code generation tasks, the PPOCoder (Shojaee et al., 2023) uses semantic matching of abstract syntax trees and data flow graphs. However, it still focuses on surface-level structure and does not fully capture the deep semantic information.

4 Methodology

We introduce Graph-Reward-SQL, a novel reward model framework designed to enhance SQL generation through two key innovations. First, we propose GMNScore, which replaces EX, reducing time costs while maintaining the accuracy of reward signals without requiring database execution. Second, we introduce StepRTM, a stepwise reward model based on the Relational Operator Tree (ROT) representation of CTE SQL, which provides supplementary intermediate feedback.

4.1 Relational Operator Tree (ROT)

Accurately modeling SQL structure and semantics is crucial for query analysis and comparison. SQL queries can be converted into Abstract Syntax Trees (ASTs) to capture their syntactic structure. However, unlike general programming languages, SQL lacks key representations like Control Flow Graphs (CFGs) (Cota et al., 1994) and Data Flow Graphs (DFGs) (Orailoglu and Gajski, 1986), which are essential for reflecting logic and data dependencies.

To bridge this gap, we leverage the Relational Operator Tree (ROT) to represent SQL queries as trees of relational algebra operators. Each node in the tree corresponds to a specific logical operation (e.g., `Join`, `Project`, `Filter`), while the tree structure itself reflects the dependencies and execution order of the query. In practice, we use Apache Calcite (Begoli et al., 2018) to generate ROTs, which compiles SQL into a canonical intermediate representation called `RelNode`. This format includes various optimizations, such as operator reordering and clause simplification, resulting in normalized logical plans that are more resilient to surface-level differences. Similar to CFGs and DFGs, the RelNode format can also integrate control dependencies and data flow as edges (Zhan et al., 2025). This enables the creation of more comprehensive graph representations that include richer SQL semantics, facilitating query understanding and evaluation.

4.2 FuncEvalGMN

After obtaining the SQL graph representations G_1 and G_2, we employ a Graph Matching Network (GMN) (Li et al., 2019) trained in SQL pairs (Zhan et al., 2025) to assess functional equivalence. It is trained using contrastive learning for pretraining and supervised learning to capture deep semantic similarity of SQL queries. The similarity between two queries is computed as the negative Euclidean distance between their final graph-level embeddings: $s(h_{G_1}, h_{G_2}) = \left\| h_{G_1} - h_{G_2} \right\|_2$, where h_{G_1} and h_{G_2} are computed by the GMN, considering the joint representations of G_1 and G_2. This approach, first introduced in FuncEvalGMN (Zhan et al., 2025), is described in further detail in Appendix O, and the details of our further optimization are provided in Appendix E.

4.3 ROT/RelNode Partial Matching (RelPM)

Similar to AST, RelNode can also be used to evaluate SQL similarity through graph matching. RelPM (Zhan et al., 2025) is a rule-based matching algorithm that assesses the similarity of SQLs based on their RelNode representations, denoted $\mathcal{G}_{\hat{q}}$ and $\mathcal{G}_{q^\star}$, respectively. A comparable approach, applied to AST structures, is known as AstPM (Zhan et al., 2025). Both algorithms adopt a hierarchical partial matching strategy and derive a global similarity score based on the Precision and Recall of node-level matching results. At the node level, matches are determined by comparing each generated node $n' \in \mathcal{G}_{\hat{q}}$ with all the candidate nodes $n \in \mathcal{G}_{q^\star}$ in the reference tree. A match is established when two nodes have the same operator type and value. Additionally, a matching score is computed by comparing their subgraphs, and the candidate node with the highest matching score is selected as the final match. Further details are provided in Appendix N.

4.4 Reward Function Design

Figure 1 illustrates our reward design, comprising the outcome reward model GMNScore and the stepwise model StepRTM. Given the generated SQL $\hat{q}$ and the reference SQL $q^\star$, the reward at time-step t for a sequence of length T is computed as follows:

$$\begin{aligned}\mathcal{R}_t(\hat{q}, q^*) = \; & \mathbb{1}(\mathit{cond}_{\text{eos}}) \cdot [R_{\text{GMNScore}}(\hat{q}, q^*) - \beta R_{\text{kl}}(\hat{q}_{<t})] \\ & + \mathbb{1}(\mathit{cond}_{\text{sub}}) \cdot \left[R_{\text{StepRTM}}(\hat{q}_{\leq t}, q^*) - \beta R_{\text{kl}}(\hat{q}_{<t})\right] \\ & + \mathbb{1}(\neg \mathit{cond}_{\text{eos}}) \cdot \mathbb{1}(\neg \mathit{cond}_{\text{sub}}) \cdot [-\beta R_{\text{kl}}(\hat{q}_{<t})]\,,\end{aligned}$$

where $\mathit{cond}_{\text{eos}}$ indicates the end of generation, at which point the outcome reward model R_{GMNScore} is applied. $\mathit{cond}_{\text{sub}}$ signifies the completion of a subquery, triggering the stepwise reward model R_{StepRTM} to compare the current subquery with the corresponding substructure in the reference query. The symbol $\neg$ denotes logical negation. $R_{\text{kl}}(\hat{q}_{<t})$ represents a KL-divergence penalty that measures the deviation between the learned policy and the pretrained language model, applied at each time step to regularize policy updates. The scalar β is a hyperparameter that balances rewards with policy regularization.

4.5 Outcome Reward: GMNScore

As described in Section 4.2, the functional correctness of generated SQL can be evaluated using the FuncEvalGMN metric $\mathcal{M}_{\text{GMN}}$, which aligns well with the objective of reward model in RL. We design an outcome reward model as follows:

$$R_{\text{GMNScore}}(\hat{q}, q^\star) = \begin{cases} -1, & \text{if syntax error} \\ -0.6, & \text{if ROT error} \\ \max(0, \mathcal{M}_{\text{GMN}} + 1) & \end{cases}$$

The GMNScore formulation introduces graded penalties for SQL queries that trigger syntax errors or ROT parsing errors[1]. For all other cases, we rescale the similarity metric $\mathcal{M}_{\text{GMN}}$ (which lies in the range $(-\infty, 0]$) to the interval $[0, 1)$ by first applying an affine shift and then rectifying any negative values to zero.

4.6 Stepwise Reward: StepRTM

Current ETL (Extract, Transform, Load) pipelines rarely execute their logic in a single step. Instead, analysts break the workflow into a detailed plan of subqueries, where each subquery progressively transforms the data until the query is complete. This logic is typically expressed using CTEs, as demonstrated by the simplified example below:

```
WITH step1 AS (/* subquery1 */),
     step2 AS (/* subquery2 */)
SELECT ... FROM step2;
```

In most cases, CTEs enhance the readability of complex SQL by providing clear representations of intermediate steps in an ETL pipeline. These steps not only facilitate data transformation but also offer a natural way to evaluate the process stepwise.

Inspired by subgraph matching techniques (Lou et al., 2020; Roy et al., 2022), we propose Stepwise Relational Operator Tree Matching (StepRTM),

[1]Refers to failures in converting SQL into a ROT using Apache Calcite. For example, the query `WITH sub1 AS (SELECT id, name FROM users) SELECT age FROM sub1;` leads to an error "Column 'age' not found in any table".

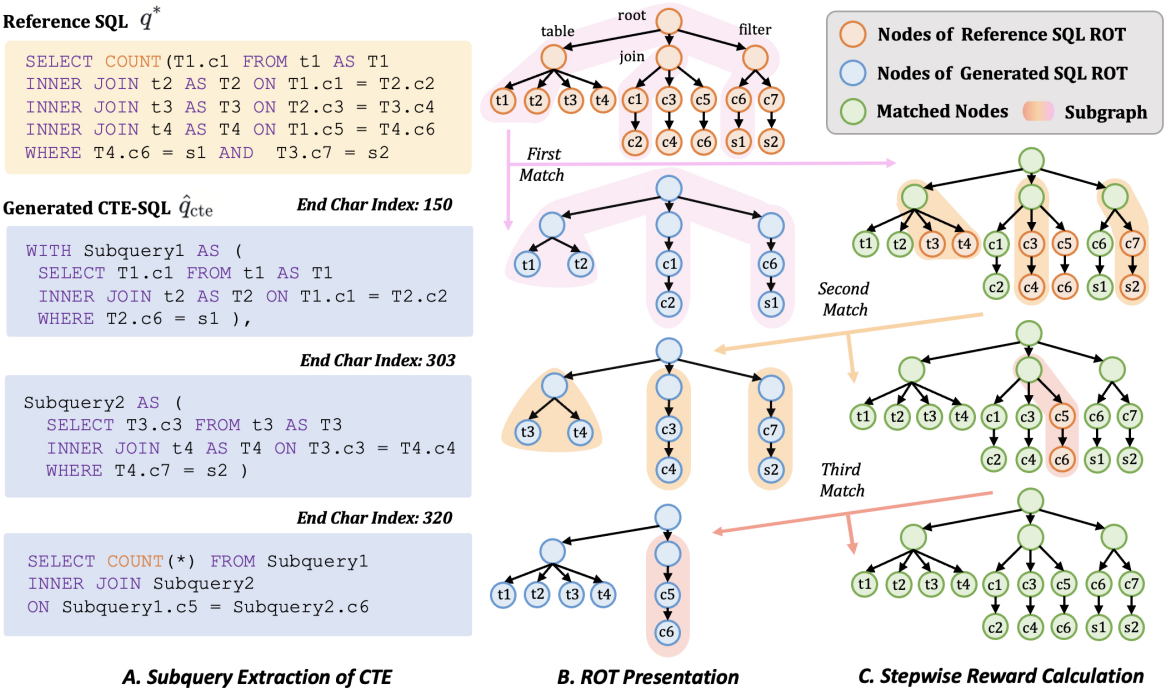

Figure 2: Overview of the StepRTM Stepwise Reward Calculation. (a) The generated SQL $\hat{q}_{\text{cte}}$ is segmented into a sequence of subqueries, with the end index of each subquery recorded. (b) Both the reference SQL query q^* and each subquery are parsed into ROTs (c) A stepwise matching process is performed between the ROTs. At each step, newly matched nodes are identified and used to compute incremental rewards.

which incorporates stepwise reward scores to provide intermediate feedback. The overall procedure of StepRTM is illustrated in Figure 2. Let q^* denote the reference SQL, and represent the generated SQL as a sequence of subqueries $\hat{q}_{\text{cte}} = [\hat{q}_1, \hat{q}_2, \ldots, \hat{q}_n]$. Let $\mathcal{G}_{q^*}$ and $\mathcal{G}_{\hat{q}_i}$ denote the node sets of the ROT representations for the reference query and the i-th generated subquery. The stepwise scores are then computed as follows:

$$\mathcal{R}^{(i)}_{\text{StepRTM}}(\hat{q}_{\text{cte}}, q^*) = \frac{\left|(\mathcal{M}_i \cup \mathcal{G}_i) \cap \mathcal{G}_{q^*}\right|}{|\mathcal{G}_{q^*}|},$$

where $\mathcal{M}_i = \bigcup_{j=1}^{i-1} \mathcal{G}_j$ represents all the matched subgraphs parsed from the first i subqueries, $\mathcal{G}_j$ denotes the maximal matched subgraph in the reference query that aligns with the i-th subquery $\hat{q}_i$. This formulation prevents repeated rewards for the same reference node and ensures that the overall signal reflects the incremental semantic coverage of the target reference query. Stepwise supervision improves training performance by providing richer intermediate feedback, facilitating the generation of correct SQL queries.

5 Experimental Setup

Datasets. Our experiments are primarily conducted on the Spider and BIRD benchmarks. The Spider dataset (Yu et al., 2018) contains 10,181 natural language questions paired with 5,693 complex SQL queries across 138 domains. The BIRD dataset (Li et al., 2024b) consists of 12,751 questions spanning more than 37 professional fields. We used the training split of the Spider dataset for training and the development splits of Spider and BIRD for evaluation. Additionally, we used a subset of the 200k-Text2SQL dataset[2] in a warm-up phase prior to RL training. Further details about the datasets are provided in Appendix A.

Baselines. We benchmark against representative state-of-the-art (SOTA) Text-to-SQL pipelines spanning in-context learning, self-correction, post-training, and multi-agent paradigms (Pourreza and Rafiei, 2023; Gao et al., 2024a; Pourreza and Rafiei, 2024; Gao et al., 2024b; Pourreza et al., 2025a,b; Zhang et al., 2025). We further compare three widely adopted reward model designs in RL training to validate the effectiveness of the proposed reward models: (i) the execution-based reward EX, widely used in recent studies (Nguyen et al., 2025; Berdnyk and Collery, 2025; Ma et al., 2025; Pourreza et al., 2025b); (ii) an LLM-based Bradley–Terry reward model (BTRM) (Christiano

[2] https://huggingface.co/datasets/philikai/200k-Text2SQL

et al., 2017; Zeng et al., 2025b), trained using the DeepSeek-Coder-1.3B-Ins as the backbone, as detailed in Appendix M, to evaluate the efficacy of model-based reward mechanisms; and (iii) AstPM and RelPM (Zhan et al., 2025), which, motivated by recent works (Shojaee et al., 2023; Nguyen et al., 2025; Pourreza et al., 2025b), are incorporated as matching-based reward model baselines.

Evaluation Metrics. Consistent with prior work (Pourreza et al., 2025a,b; Zhang et al., 2025), we adopt Execution Accuracy (EX) as the primary metric for comparisons with state-of-the-art systems. In addition, we report Test-Suite Accuracy (TS), which evaluates predicted queries against suites of distilled or fuzzed database variants under varied conditions (Zhong et al., 2020). Since TS mitigates the false positives that may arise from insufficiently discriminative databases, it provides a more robust estimate of semantic correctness (Gao et al., 2024a; Li et al., 2024a; Yang et al., 2024). We therefore use TS to more accurately quantify the gains achieved by our reward model.

Implementation Details. All experiments are conducted on several edge servers. We extend the verl[3] and DeepSpeed-Chat[4] (Yao et al., 2023) framework to support a comparison of multiple reward models, including execution-based, LLM-based, and matching-based rewards.

Prior to RL training, we performed SFT using two cold-start datasets. First, we sampled a subset from the 200k-Text2SQL dataset, matching the size of the Spider training set, and trained the base model for two epochs. To promote the generation of CTE SQL queries in the stepwise reward PPO experiments, we converted BIRD data into CTE format to prepare a warm-up dataset referred to as CTE-SFT. During supervised fine-tuning (SFT), we employ the AdamW optimizer (initial learning rate 5e-6, batch size 4) and use a polynomial scheduler to decay the learning rate throughout training. Additional details about hyperparameter of SFT and RL are provided in Appendix C.

6 Results

We benchmark our method against recent SOTA Text-to-SQL systems, including in-context learning and multi-agent approaches. As shown in Table 2, Graph-Reward-SQL reaches 81.62% EX on Spider and 63.04% on BIRD trained with GRPO on the Qwen2.5-Coder-7B-Ins model. Our reward model framework delivers performance competitive with the Reasoning-SQL, even as the latter employs a multi-agent system and five reward models that incorporate both EX and LLM-based judgment.

6.1 Reward Performance Comparison

GMNScore can replace the EX, thereby eliminating dependence on SQL execution and database environments. As demonstrated in Table 3, GMNScore achieves the highest average TS for the 1.3B and 6.7B models, highlighting the importance of well-designed reward signals in RL. Another notable observation is that RelPM outperforms AstPM, with improvements of 2.53% and 1.71% for the two model sizes, respectively. The better performance of the former can be attributed to the use of normalized logical plans for SQL parsing in ROT, which are less susceptible to surface-level syntactic differences. ROT also provides an effective representation way for our proposed GMNScore and StepRTM reward models. GMNScore learns deep semantic information via graph-level embeddings, bypassing the need for execution-result comparisons and thus mitigating false-positive noise. Additionally, GMNScore eliminates the necessity of constructing and maintaining databases, offering a lightweight solution for large-scale RL-based Text-to-SQL. Case studies are provided in Appendix S.

The integration of StepRTM as a stepwise reward further enhances performance. As shown in Table 4, combining CTE-SFT with StepRTM consistently results in metric improvements across various outcome reward models. Notably, our framework, which integrates GMNScore alongside StepRTM, achieves the highest result. Specifically, we observe a 5.87% improvement on the BIRD dataset and a 0.97% increase on the Spider dataset. These observations suggest that the BIRD dataset, which is inherently more challenging due to its diverse database and query complexity, benefits more significantly from our proposed stepwise reward.

6.2 Effectiveness of GMNScore with GRPO

GMNScore performs robustly across both PPO and GRPO protocols. As shown in Figure 3, the results consistently demonstrate that GMNScore outperforms EX in these two RL protocols, underscoring its robustness and effectiveness. We report the average performance across two benchmarks,

[3] https://github.com/volcengine/verl
[4] https://github.com/deepspeedai/DeepSpeed

Method	Method Detail	Base Model	Spider EX	BIRD EX	Avg.
DIN-SQL (Pourreza and Rafiei, 2023)	In-context learning	GPT-4	82.88	50.72	66.80
DAIL-SQL (Gao et al., 2024a)	In-context learning	GPT-4	74.47	54.76	64.62
DTS-SQL (Pourreza and Rafiei, 2024)	Schema-linking, SFT	DeepSeek-7B	-	55.80	-
XiYan-SQL (Gao et al., 2024b)	Multi-Agent	QwenCoder-32B	-	67.01	-
CHASE-SQL (Pourreza et al., 2025a)	Multi-Agent	Gemini-1.5-pro	-	74.46	-
Reasoning-SQL (Pourreza et al., 2025b)	Multi-Agent, GRPO	Qwen2.5-Coder-7B	78.72	**64.01**	71.37
Reward-SQL (Zhang et al., 2025)	GRPO	Qwen2.5-Coder-7B	77.08	59.70	68.39
Graph-Reward-SQL (Ours)	GRPO	Qwen2.5-Coder-7B	**81.62**	63.04	**72.33**

Table 2: Performance comparison of Graph-Reward-SQL and baseline models on the Spider and BIRD dataset. Baseline results are reported from their original publications.

Method	Spider	BIRD	Avg.
DeepSeek-Coder-1.3B-Ins	39.56	11.34	25.45
+ SFT	57.74	13.30	35.52
+ PPO w/ AstPM	59.96	12.52	36.24
+ PPO w/ RelPM	62.86	14.67	38.77
+ PPO w/ BTRM	61.41	14.21	37.81
+ PPO w/ EX	65.28	**17.21**	41.25
+ PPO w/ GMNScore (Ours)	**67.70**	16.10	**41.90**
DeepSeek-Coder-6.7B-Ins	44.97	18.38	31.68
+ SFT	69.05	21.71	45.38
+ PPO w/ AstPM	70.31	22.88	46.60
+ PPO w/ RelPM	70.60	26.01	48.31
+ PPO w/ BTRM	67.99	22.23	45.11
+ PPO w/ EX	71.66	**26.66**	49.16
+ PPO w/ GMNScore (Ours)	**72.44**	26.14	**49.29**

Table 3: TS Performance of Deepseek-Coder-1.3B-Ins and Deepseek-Coder-6.7B-Ins models under multiple baselines and proposed GMNScore outcome reward.

Method	Spider	BIRD	Avg.
DeepSeek-Coder-1.3B-Ins + PPO	39.56	11.34	25.45
RelPM w/ SFT	62.86	14.67	38.77
RelPM w/ CTE-SFT	62.57	19.36	40.97
RelPM & StepRTM w/ CTE-SFT	**64.31**	**19.43**	**41.87**
EX w/ SFT	65.28	17.21	41.25
EX w/ CTE-SFT	65.09	18.97	42.03
EX & StepRTM w/ CTE-SFT	**67.89**	**19.75**	**43.82**
GMNScore w/ SFT	67.70	16.10	41.90
GMNScore w/ CTE-SFT	68.57	20.40	44.49
GMNScore & StepRTM w/ CTE-SFT	**68.67**	**21.97**	**45.32**

Table 4: TS performance of the DeepSeek-Coder-1.3B-Ins model trained with StepRTM integration. Both SFT and CTE-SFT refer to different warm-up datasets.

with detailed results provided in Appendix K.

6.3 Cost of GMNScore

We compare the cost of different reward models by running one PPO epoch and measuring the average reward-scoring latency per sample. As shown in Table 5, GMNScore is the most efficient. In contrast, EX is the slowest due to repeated database calls. From a model-size perspective, GMNScore is

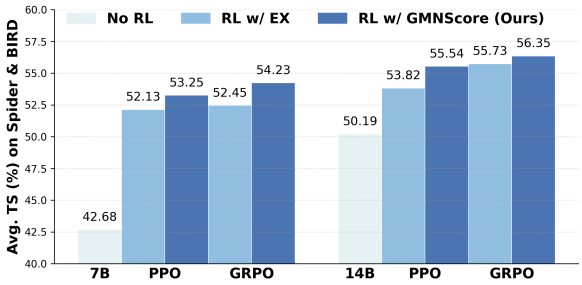

Figure 3: TS Performance of Qwen2.5-Coder-7B/14B-Ins models directly trained by PPO/GRPO.

Reward	Time↓	Params↓	GPU Memory↓
EX	1.088s	–	–
BTRM	0.095s	1.35B	9304MB
GMNScore	**0.023s**	**3.99M**	**83MB**

Table 5: Average per-sample cost of reward scoring.

highly compact, with 3.99M parameters, whereas the widely used BTRM is substantially larger at 1.35B parameters. GMNScore is trained once offline and can be reused across multiple RL experiments, which significantly reduces overall computational overhead. Additional details on the measurement setup are provided in Appendix G.

6.4 Intrinsic Evaluation of Reward Models

Area Under the Curve (AUC) is used as a statistically principled measure of similarity-evaluation accuracy (Zhan et al., 2025); its alignment with RL reward-model objectives makes it a rigorous criterion for validating reward-model performance. As shown in Table 7, GMNScore achieves 97.62% AUC on Spider-dev-pair and 94.14% on BIRD-dev-pair. Equivalence labels in the these two dev-pair sets were manually verified to minimize false positives, improving the reliability of these AUC estimates. Collectively, the results indicate that GMNScore provides accurate and consistent preference judgments, laying a solid foundation for subsequent RL training. Details on training and

Reference SQL	Failed SQL ✗	CTE SQL ✓
Question: What is the average score of Stephen Turner's posts?		
`SELECT AVG(T2.Score) FROM users AS T1 INNER JOIN posts AS T2 ON T1.Id = T2.OwnerUserId WHERE T1.DisplayName = 'Stephen Turner'`	`SELECT AVG(T1.Score) FROM comments AS T1 INNER JOIN users AS T2 ON T1.UserID = T2.ID WHERE T2.DisplayName = 'Stephen Turner'`	`WITH UserInfo AS ( SELECT id FROM users WHERE displayname = 'Stephen Turner'), PostInfo AS ( SELECT score FROM posts WHERE owneruserid IN (SELECT id FROM UserInfo)) SELECT AVG(score) FROM PostInfo`
Question: List down at least five superpowers of male superheroes.		
`SELECT T3.power_name FROM superhero AS T1 INNER JOIN hero_power AS T2 ON T1.id = T2.hero_id INNER JOIN superpower AS T3 ON T3.id = T2.power_id INNER JOIN gender AS T4 ON T4.id = T1.gender_id WHERE T4.gender = 'Male' LIMIT 5`	`SELECT T1.power_name FROM superpower AS T1 INNER JOIN hero_power AS T2 ON T1.id = T2.power_id INNER JOIN superhero AS T3 ON T3.id = T2.hero_id WHERE T3.gender_id = 1 GROUP BY T1.power_name LIMIT 5`	`WITH MaleSuperheroes AS (SELECT id FROM superhero WHERE gender_id IN (SELECT id FROM gender WHERE gender = 'Male'), SuperpowersOfMaleSuperheroes AS (SELECT DISTINCT T1.power_name FROM superpower AS T1 INNER JOIN hero_power AS T2 ON T1.id = T2.power_id INNER JOIN MaleSuperheroes AS T3 ON T3.id = T2.hero_id) SELECT power_name FROM SuperpowersOfMaleSuperheroes LIMIT 5`

Table 6: Comparisons among Reference SQL, Failed SQL, and CTE SQL demonstrate the effectiveness of StepRTM.

Reward Model	Spider-pair dev	BIRD-pair dev
AstPM	82.81%	80.38%
RelPM	84.42%	83.57%
BTRM	89.24%	84.77%
EX	96.37%	92.67%
GMNScore	**97.62%**	**94.14%**

Table 7: Area Under the Curve (AUC) of reward models on Spider-dev-pair and BIRD-dev-pair.

evaluation datasets are provided in Appendix E.

6.5 Case Study: CTE SQL with StepRTM

Table 6 presents two cases that demonstrate how the stepwise reward model enhances both correctness and readability. Each case compares the reference SQL, a failed SQL query generated by a model trained solely with an outcome reward model, and a CTE SQL query generated by a model trained with StepRTM. In the first case, the failed SQL incorrectly retrieves data from the '`comments`' table instead of the intended '`posts`' table. The CTE SQL resolves this by decomposing the task into clear subqueries: first locating the target user, then aggregating the scores of that user's posts. In the second case, the failed SQL hard-codes the gender identifier as '1', leading to errors in filtering. In contrast, the CTE SQL uses two dedicated subqueries to correctly filter male superheroes, extract their superpowers, finally combine the results.

7 Discussion

The GMNScore introduced in this paper offers an alternative to EX while remaining fully compatible with other reward models. As detailed in Appendix I, we extend our investigation beyond the StepRTM integration (Section 6.1) by applying hybrid outcome reward models, which further improve performance. This finding is consistent with previous work using multiple outcome rewards (Pourreza et al., 2025b; Ma et al., 2025).

8 Conclusion

We propose Graph-Reward-SQL, a reward model framework for RL-based Text-to-SQL that replaces execution-based rewards with the GMNScore outcome reward model and the StepRTM stepwise reward model. By eliminating the dependency on execution during training, it significantly improves training efficiency. Extensive experiments demonstrate that our proposed reward models achieve superior performance. The Graph-Reward-SQL framework establishes a new direction toward scalable and efficient RL-based Text-to-SQL tasks.

9 Limitations

While our proposed reward models demonstrate strong performance in text-to-SQL tasks and show much better performance than the execution-based reward model, its SQL-specific architectural design currently restricts its applicability to broader code generation domains. Generalizing our reward model framework to support programming languages such as Python and Java would require some modifications to the language representation mechanisms, particularly to address the challenge of assessing functional equivalence across diverse programming paradigms.

However, this limitation opens promising directions for future work. We aim to explore the applicability of the execution-free reward model in broader code generation tasks in future work, thereby offering viable alternatives to the common RL practice of relying solely on execution outcomes. This extension may contribute to more general, efficient, and flexible reinforcement learning training for code generation tasks.

References

Edmon Begoli, Jesús Camacho-Rodríguez, Julian Hyde, Michael J Mior, and Daniel Lemire. 2018. Apache calcite: A foundational framework for optimized query processing over heterogeneous data sources. In *Proceedings of the 2018 International Conference on Management of Data*, pages 221–230.

Mariia Berdnyk and Marine Collery. 2025. Llm-based sql generation with reinforcement learning. In *The First Workshop on Neural Reasoning and Mathematical Discovery at AAAI'2025.* Workshop Paper.

Ralph Allan Bradley and Milton E Terry. 1952. Rank analysis of incomplete block designs: I. the method of paired comparisons. *Biometrika*, 39(3/4):324–345.

Paul F. Christiano, Jan Leike, Tom B. Brown, Miljan Martic, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, page 4302–4310, Red Hook, NY, USA. Curran Associates Inc.

Bruce A Cota, Douglas G Fritz, and Robert G Sargent. 1994. Control flow graphs as a representation language. In *Proceedings of Winter Simulation Conference*, pages 555–559. IEEE.

Richard Cyganiak. 2005. A relational algebra for SPARQL. *Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170*, 35(9).

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

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

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, et al. 2024b. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming - the rise of code intelligence. *ArXiv*, abs/2401.14196.

Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards complex text-to-SQL in cross-domain database with intermediate representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 4524–4535, Florence, Italy. Association for Computational Linguistics.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Kai Dang, An Yang, Rui Men, Fei Huang, Shanghaoran Quan, Xingzhang Ren, Xuancheng Ren, Jingren Zhou, and Junyang Lin. 2024. Qwen2.5-coder technical report. *ArXiv*, abs/2409.12186.

Hung Le, Yue Wang, Akhilesh Deepak Gotmare, Silvio Savarese, and Steven Chu Hong Hoi. 2022. Coderl: Mastering code generation through pretrained models and deep reinforcement learning. *Advances in Neural Information Processing Systems*, 35:21314–21328.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: Leveraging multiple prompts and multiple-choice selection for text-to-SQL generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353, Abu Dhabi, UAE. Association for Computational Linguistics.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, et al. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the role of schema linking in text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6943–6954, Online. Association for Computational Linguistics.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024a. 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, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. 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.

Yujia Li, Chenjie Gu, Thomas Dullien, Oriol Vinyals, and Pushmeet Kohli. 2019. Graph matching networks for learning the similarity of graph structured objects. In *International conference on machine learning*, pages 3835–3845. PMLR.

Yujia Li, Daniel Tarlow, Marc Brockschmidt, and Richard Zemel. 2015. Gated graph sequence neural networks. *arXiv preprint arXiv:1511.05493*.

Zhaoyu Lou, Jiaxuan You, Chengtao Wen, Arquimedes Canedo, Jure Leskovec, et al. 2020. Neural subgraph matching. *arXiv preprint arXiv:2007.03092*.

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

Xuan-Bang Nguyen, Xuan-Hieu Phan, and Massimo Piccardi. 2025. Fine-tuning text-to-sql models with reinforcement-learning training objectives. *Natural Language Processing Journal*, 10:100135.

Alex Orailoglu and Daniel D Gajski. 1986. Flow graph representation. In *Proceedings of the 23rd ACM/IEEE Design Automation Conference*, pages 503–509.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025a. CHASE-SQL: Multi-path reasoning and preference optimized candidate selection in text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

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

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.

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

Indradyumna Roy, Venkata Sai Baba Reddy Velugoti, Soumen Chakrabarti, and Abir De. 2022. Interpretable neural subgraph matching for graph retrieval. *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(7):8115–8123.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Y Wu, et al. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Jie Shi, Bo Xu, Jiaqing Liang, Yanghua Xiao, Jia Chen, Chenhao Xie, Peng Wang, and Wei Wang. 2025. Gen-SQL: Efficient text-to-SQL by bridging natural language question and database schema with pseudo-schema. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 3794–3807, Abu Dhabi, UAE. Association for Computational Linguistics.

Parshin Shojaee, Aneesh Jain, Sindhu Tipirneni, and Chandan K. Reddy. 2023. Execution-based code generation using deep reinforcement learning. *Transactions on Machine Learning Research*.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393, Singapore. Association for Computational Linguistics.

Anna L Trella, Kelly W Zhang, Inbal Nahum-Shani, Vivek Shetty, Finale Doshi-Velez, and Susan A Murphy. 2023. Reward design for an online reinforcement learning algorithm supporting oral self-care. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 15724–15730.

Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-aware schema encoding and linking for text-to-SQL parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7567–7578, Online. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. Synthesizing text-to-SQL data from weak and strong LLMs. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7864–7875, Bangkok, Thailand. Association for Computational Linguistics.

Zhewei Yao, Reza Yazdani Aminabadi, Olatunji Ruwase, Samyam Rajbhandari, Xiaoxia Wu, Ammar Ahmad Awan, Jeff Rasley, Minjia Zhang, Conglong Li, Connor Holmes, Zhongzhu Zhou, Michael Wyatt, Molly Smith, Lev Kurilenko, Heyang Qin, Masahiro Tanaka, Shuai Che, Shuaiwen Leon Song, and Yuxiong He. 2023. DeepSpeed-Chat: Easy, Fast and Affordable RLHF Training of ChatGPT-like Models at All Scales. *arXiv preprint arXiv:2308.01320*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir

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

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. 2025a. Acecoder: Acing coder rl via automated test-case synthesis. *ArXiv*, 2502.01718.

Huaye Zeng, Dongfu Jiang, Haozhe Wang, Ping Nie, Xiaotong Chen, and Wenhu Chen. 2025b. Acecoder: Acing coder rl via automated test-case synthesis.

Yi Zhan, Longjie Cui, Han Weng, Guifeng Wang, Yu Tian, Boyi Liu, Yingxiang Yang, Xiaoming Yin, Jiajun Xie, and Yang Sun. 2025. Towards database-free text-to-SQL evaluation: A graph-based metric for functional correctness. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 4586–4610, Abu Dhabi, UAE. Association for Computational Linguistics.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-sql: Boosting text-to-sql via stepwise reasoning and process-supervised rewards. *ArXiv*, abs/2505.04671.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 396–411, Online. Association for Computational Linguistics.

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

A Datasets Details

Spider (Yu et al., 2018) is a large-scale, cross-domain Text-to-SQL benchmark containing 10,181 questions and 5,693 complex SQL queries. These queries come from 200 multi-table databases across 138 domains. The dataset is split into three subsets without any overlap in databases: training (8659), development (1034) and test datasets. This separation helps ensure a fair assessment of model performance.

BIRD (Li et al., 2024b) is another large-scale and cross-domain Text-to-SQL dataset known for its complexity. It contains 12,751 question-SQL pairs drawn from 95 databases across more than 37 professional fields, making it a challenging testbed for evaluating the generalization capability of semantic parsers. A notable characteristic of BIRD is its "dirty" nature: queries may be inaccurate, column names are often ambiguous or poorly described, and databases may contain null values or irregular encodings. These issues collectively pose unique challenges for model robustness. The dataset is split into 9,428 training examples and 1,534 development examples, with the remaining reserved for testing.

200k-Text2SQL comprises 200,000 examples related to the text-to-SQL task. Each data entry also includes a question, a database schema, and a reference SQL statement. To prepare the model for generating clean SQL queries during the PPO phase, we first fine-tune the Deepseek-coder-1.3b-ins model via Supervised Fine-Tuning (SFT), which enables the model to follow instructions and complete Text-to-SQL tasks effectively. The prompt used is detailed in Q. The dataset is available at: `https://huggingface.co/datasets/philikai/200k-Text2SQL`.

B Evaluation Setting Details

Test-Suite (TS) Performance provides a robust and reliable assessment of semantic accuracy. For example, the original `academic.sqlite` database is automatically augmented into 411 variants, resulting in a total of 412 test databases. Each predicted-reference SQL pair is executed on all of them. This reduces the likelihood of false positives and strengthens the evaluation's ability to detect semantic discrepancies.

In our evaluation using the official TS framework, we emphasize strictness in semantic verification. Specifically, we enable the `-keep_distinct` flag to preserve the use of the `DISTINCT` keyword during evaluation. Meanwhile, we disable `-plug_value` to avoid injecting ground-truth values into predicted queries, thus requiring the model to generate complete SQL outputs, including correct values. We use the TS code to evaluate Spider on its augmented dev split databases. For BIRD, since no augmented databases are provided, we apply the TS code directly on its original database. The code and augmented databases are available at: `https://github.com/taoyds/test-suite-sql-eval`.

In our experiments using the TS evaluation metric, all hints (e.g., age = year - birth_year) in BIRD dataset were deliberately omitted to align with the setup of the Spider dataset. This methodological choice results in lower performance metrics compared to studies that incorporated the hints available in the BIRD dataset.

C Implement Details

C.1 Outcome Reward Experiments

For the outcome reward PPO experiments, we first conduct a warm-up phase using a different dataset from the target training corpus—200k-Text2SQL. We sample an equivalent number of examples as in the training split of Spider dataset and perform 2 epochs of SFT warm-up using SFT. The sampling process also involves removing irrelevant samples, such as those with queries in Chinese, to ensure consistency and relevance in the training data. This warm-up helps the model adapt to prompt-following behavior and prevents it from generating non-SQL text. After SFT warm-up, PPO training proceeds on the Spider/BIRD dataset with hyperparameters shown in table 8. These hyperparameters, along with those for our reward model, were either adopted from prior work (Shojaee et al., 2023) or determined through empirical experiments.

C.2 Stepwise Reward Experiments

For the stepwise reward PPO experiments, we encourage the generation of CTE-structured SQLs by incorporating CTE-rewritten data into the SFT stage. Since SQL queries in BIRD are generally more complex than those in Spider, they are particularly well-suited for decomposition into modular subqueries using the CTE format. To leverage this, we rewrite the reference SQLs in the BIRD train-

Table 8: PPO hyperparameters.

Hyperparameter	Default Value
Optimizer	AdamW
Learning Rate	5e-6
Scheduler	Constant
Max Length	1024
Batch Size	32
N_{mb} Number of Mini-batches	8
Gradient Accumulation Steps	8
β (KL Penalty Coefficient)	0.05
γ (Discount Factor)	0.99
λ (for GAE)	0.99
K (PPO Update Iteration)	1
ε (PPO's Policy Clipping Coefficient)	0.1
Value Function Loss Clipping	True
$\hat{\varepsilon}$ (Value Clipping Coefficient)	0.2
Sampling Temperature	1

ing set into CTE format using GPT-4o, following the prompt strategies outlined in Appendix Q. The resulting data is then combined with the warm-up dataset used in the outcome reward model experiments to construct a SFT warm-up set. This setup encourages the model to generate CTE SQL, enabling effective application of stepwise reward signals during PPO training.

C.3 Qwen2.5-Coder-7B-Ins Experiments

Given the stronger capabilities of the base model, we directly applied reinforcement learning to the Qwen2.5-Coder-7B-Ins model using two distinct RL protocols, GRPO and PPO, without any warm-up training.

In experiments of Group Relative Policy Optimization (GRPO) with Qwen2.5-Coder-7B-Ins, we configured the learning rates for the actor and critic to 5e-6 and 2e-6, respectively, with a warm-up period of 10 steps to ensure stable training dynamics in the early stages. The model was trained with a batch size of 512, and mini-batches of size 256 were used. A weight decay of 0.1 was applied to regularize the model and mitigate overfitting. The group size for GRPO was set to 16, which corresponds to the number of completions generated per input prompt, thereby facilitating the optimization of the policy through relative comparisons between groups. For effective handling of long sequences, we set the maximum prompt length to 4096 and the maximum response length to 16,384.

We visualized the reward scores and entropy values of the training dataset during the training process, as shown in Figure 4 and Figure 5.

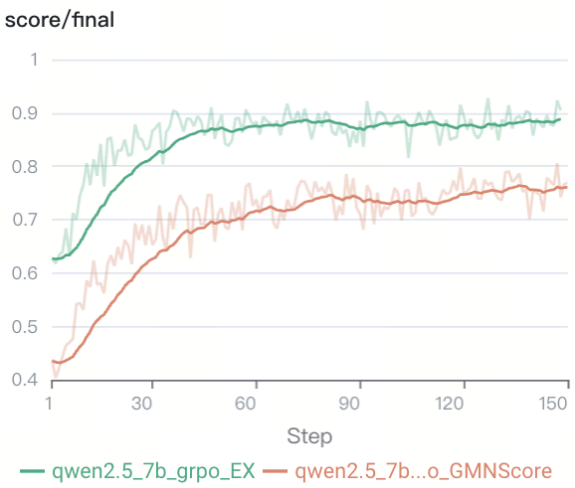

Figure 4: Reward score trends on the training set during GRPO training. The green curve corresponds to the execution-based reward model (EX), while the orange curve shows the results using the proposed reward model GMNScore. Both models are trained with the Qwen2.5-Coder-7B-Ins backbone under the GRPO framework. While GMNScore starts from a lower initial score compared to EX, it demonstrates a larger overall increase.

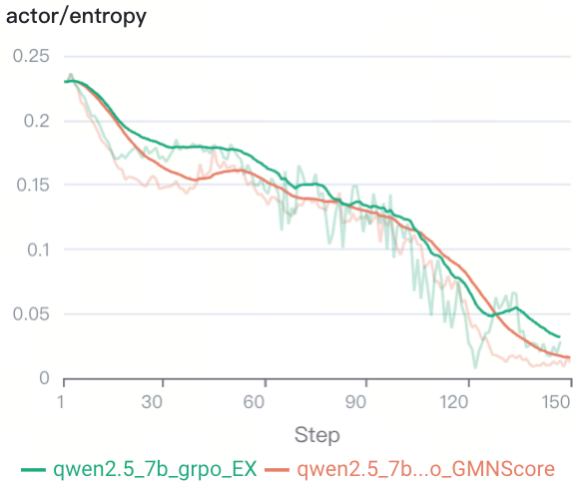

Figure 5: Actor entropy of GRPO training. Both models are based on Qwen2.5-Coder-7B-Ins. The orange curve corresponds to GMNScore and the green curve to EX. Entropy consistently decreases as the policy becomes more deterministic. GMNScore exhibits a faster reduction in entropy during the early training stages, while both methods converge to similarly low-entropy levels by the end of training.

D GRPO

In addition to experiments based on PPO, we also evaluate our methods under the Group Relative Policy Optimization (GRPO) RL framework (Shao et al., 2024), in order to further assess the effectiveness of our proposed GMNScore outcome

reward model. In the standard text-to-SQL setting, let x denotes a natural language query, and $\mathcal{G} = \{\hat{q}_1, \ldots, \hat{q}_G\}$ denotes a group of G candidate SQL queries sampled from the current policy $\pi_\theta(\cdot|x)$. To better leverage relative preferences among candidates, GRPO optimizes the policy by maximizing the following objective:

$$\begin{aligned}\mathcal{J}(\theta) =& \mathbb{E}_{x\sim\mathcal{D},\, \mathcal{G}\sim\pi_\theta(\cdot|x)^G} \\ & \Bigg[\frac{1}{G}\sum_{i=1}^{G} \min\left(\frac{\pi_\theta(\hat{q}_i|x)}{\pi_{\theta_{\text{old}}}(\hat{q}_i|x)} A_i,\right. \\ & \left.\text{clip}\left(\frac{\pi_\theta(\hat{q}_i|x)}{\pi_{\theta_{\text{old}}}(\hat{q}_i|x)}, 1-\epsilon, 1+\epsilon\right) A_i\right) \\ & - \beta\, \mathbb{D}_{\text{KL}}\left(\pi_\theta(\cdot \mid x) \,\|\, \pi_{\text{ref}}(\cdot \mid x)\right)\Bigg],\end{aligned}$$

where π_θ and $\pi_{\theta_{\text{old}}}$ denote the current and previous policies, respectively, and π_{ref} is a fixed reference policy used for KL regularization. The term A_i represents the advantage of candidate $\hat{q}_i$ within the sampled group, reflecting its relative quality compared to other completions. The ratio $\frac{\pi_\theta(\hat{q}_i|x)}{\pi_{\theta_{\text{old}}}(\hat{q}_i|x)}$ captures the likelihood shift under the current policy, while ϵ and β are hyperparameters that control the clipping threshold and the strength of the KL penalty, respectively.

By generating multiple candidates per input, GRPO naturally accommodates the inherent ambiguities and challenges of mapping natural language to SQL queries, ensuring that feedback is both robust and informative.

E Training of GMN of GMNscore

The Graph Matching Network (GMN) proposed serves as a foundation for assessing SQL query equivalence (Zhan et al., 2025), it was not originally designed for RL-based training. In our preliminary evaluation, reward model GMNScore built on the provided GMN model of FuncEvalGMN (Zhan et al., 2025) lagged behind that of the baseline EX under PPO experiments. We conducted a failure case analysis focusing on example where scores of GMNScore reward model disagreed with execution results. After manually checking, we identified several types of potential error situations.

To develop a GMN better suited for RL-based Text-to-SQL tasks, we retrain the model using an augmented version of the Spider-train-pair dataset introduced in (Zhan et al., 2025). The original Spider-train-pair dataset consists of 17,664 SQL query pairs annotated with binary equivalence labels. Two corresponding sets for evaluation, Spider-dev-pair and BIRD-dev-pair, contain 1,644 and 2,977 examples, respectively.

Seven augmentation strategies were crafted from the identified failure cases, adding 3,397 training pairs. The strategies are summarized in Table 9. After augmentation, the training set exceeded 20,000 SQL pairs.

F Details of Execution Accuracy (EX)

Execution Accuracy (EX) is a commonly used reward signal in text-to-SQL tasks, offering supervision based on the correctness of query execution. However, we do not adopt the naive binary EX metric that only distinguishes between success and failure. Instead, we adopt a stronger variant of execution-based reward that integrates syntax check signals and runtime diagnostics, resulting in a more fine-grained supervision signal. Motivated by prior work (Pourreza et al., 2025b; Shojaee et al., 2023), this reward formulation extends traditional binary execution accuracy with syntax awareness and serves as a strong execution-based baseline for comparison.

Given a generated SQL query $\hat{q}$ and the reference SQL $q^\star$, we compute the EX reward as:

$$r_{\text{EX}}(\hat{q}, q^\star) = \begin{cases} 1, & \text{correct execution} \\ -0.3, & \text{incorrect execution} \\ -0.6, & \text{runtime error} \\ -1, & \text{syntax error} \end{cases}$$

A runtime error refers to a situation where the SQL query is syntactically correct but fails during execution. This can occur, for example, when a non-existent table or column is referenced in the query. This reward formulation provides more informative feedback to the policy model, especially during early training stages when most queries fail due to syntax or runtime errors, avoiding undifferentiated negative signals.

For the false positive problem of execution-based rewards, Test Suite (TS) (Zhong et al., 2020) attempted to improve robustness by using a set of test cases to simulate query behavior under different data distributions. Nevertheless, as shown in (Zhan et al., 2025), both false positives (semantically incorrect queries that return the right result) and false negatives (semantically correct queries that fail under test data) persist, due to the reliance on incomplete or ambiguous database contents.

Number	Augmentation Type	Original SQL	Augmented Result
		Enhancement of equivalent cases	
137	IN Clause Replacement	**Original:** SELECT Name FROM (SELECT Name, Age FROM technician) AS t WHERE Age IN (36, 37)	**Aug:** SELECT Name FROM (SELECT Name, Age FROM technician) AS t WHERE Age = 36 OR Age = 37
		Enhancement of non-equivalent cases	
422	Column Name Perturbation (Select Clause)	**Original:** SELECT actid FROM activity	**Aug:** SELECT acti FROM activity **Aug:** SELECT ctid FROM activity
566	Keyword Replacement (AND/OR)	**Original:** SELECT * FROM Products WHERE Price >= 60 AND Price <= 120	**Aug:** SELECT * FROM Products WHERE Price >= 60 OR Price <= 120
1094	Symbol Replacement (Comparison Operator)	**Original:** SELECT * FROM (SELECT dept_name, building FROM department) AS t WHERE building > (SELECT AVG(building) FROM department)	**Aug:** SELECT * FROM (SELECT dept_name, building FROM department) AS t WHERE building >= (SELECT AVG(building) FROM department)
276	Table Source Replacement	**Original:** SELECT * FROM (SELECT name, email FROM user_profiles) AS t WHERE name LIKE '%Swift%'	**Aug:** SELECT * FROM (SELECT name, email FROM user) AS t WHERE name LIKE '%Swift%'
64	Column Name Replacement	**Original:** SELECT candidate_id FROM candidate ORDER BY candidate_id DESC LIMIT 3	**Aug:** SELECT people_id FROM candidate ORDER BY people_id DESC LIMIT 3
838	Column Removal	**Original:** SELECT circuitid, location FROM (SELECT circuitid, location, country FROM circuits) AS t WHERE country = 'fraNce' OR country = 'belGium'	**Aug:** SELECT circuitid FROM (SELECT circuitid, country FROM circuits) AS t WHERE country = 'fraNce' OR country = 'belGium'

Table 9: Examples of SQL augmentation strategies used for generating non-equivalent/equivalent SQL pairs. Each example introduces specific perturbations (e.g., column name change or logical operator replacement) that alter the semantics of the original SQL.

Furthermore, frequent SQL execution significantly increases computational overhead, becoming the major bottleneck during RL training, where query evaluation must occur at each rollout step.

G Cost Analysis of Reward Models

To evaluate the computational efficiency of different reward models, we conduct experiments on an edge server equipped with an Intel Xeon Platinum 8336C CPU (128 cores) and a total memory capacity of 2.0 TiB. Our comparison focuses exclusively on model-based reward functions. Rule-based reward methods (AstPM/RelPM outcome reward model and our proposed stepwise reward model StepRTM) are excluded from this analysis as they incur negligible memory overhead.

Reward Type	Time↓	Params↓	GPU Memory↓
EX	1.088s	N/A	N/A
BTRM	0.095s	1.35B	9304MB
GMNScore	**0.023s**	**3.99M**	**83MB**

Table 10: Cost comparison across reward models.

Time Cost. We use the training split of the Spider dataset and perform one epoch of PPO training, during which we measure the total time consumed by the reward score computation. We then record the average reward calculation time per sample, providing insights into the computational efficiency of different reward models. Both BTRM and GMNScore are implemented using bfloat16 precision for acceleration. Execution-based reward (EX) incurs the highest computational cost due to repeated database calls. While EX can leverage high concurrency when executed on the CPU to accelerate performance, it still remains slower in practice due to its reliance on database calls. In real-world scenarios, different GPU rollouts of samples are processed concurrently through the reward model, which can improve processing speed. However, both BTRM and GMNScore can also easily achieve parallelization by loading the same model onto each GPU. Therefore, the speed reflected in the table is representative of the actual performance in training.

Model Size. We also compare the model sizes based on the total number of parameters using:

```
p.numel() for p in model.parameters()
```

We train Bradley-Terry Reward Model (BTRM) using DeepSeek-Coder-1.3B-Ins as the base model, with a total of 1,346,471,936 parameters. In contrast, GMNScore adopts a lightweight Graph Matching Network with only 3,994,944 parameters.

Memory Cost. To evaluate GPU memory consumption during inference, we apply:

```
torch.cuda.reset_peak_memory_stats()
```

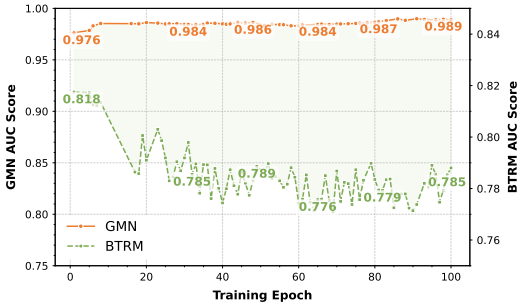

Figure 6: There is a high Area Under the Curve (AUC) score between the GMNScore and execution results, consistently exceeding 97.6% during training.

```
# here is model inference code
torch.cuda.max_memory_allocated()
```

On average, BTRM consumes approximately 9304 MB of GPU memory, while GMNScore requires only 83 MB.

H Analysis of GMNScore Accuracy

Experimental results demonstrate the effectiveness of GMNScore as a reward model in PPO, significantly outperforming BTRM. We analyze the correlation[5] between these two reward signals and actual execution outcomes during PPO training. As shown in Figure 6, GMNScore consistently maintains a high correlation with the execution results. This indicates that GMNScore provides a more stable and precise reward signal than BTRM during training, contributing to its superior performance.

I Hybrid Outcome Reward Designs

Method	Spider	BIRD	Avg.
DeepSeek-Coder-1.3B-Ins	39.56	11.34	25.45
+ PPO w/ RelPM	62.86	14.67	38.77
+ PPO w/ EX	65.28	17.21	41.25
+ PPO w/ GMNScore	67.70	16.10	41.90
+ PPO w/ EX & AstPM	62.38	16.43	39.41
+ PPO w/ GMNScore & AstPM	65.28	16.17	40.73
+ PPO w/ GMNScore & RelPM	66.34	16.69	41.52
+ PPO w/ EX & GMNScore	67.02	17.73	42.38
+ PPO w/ EX & RelPM	**68.28**	**18.77**	**43.53**

Table 11: TS Performance of DeepSeek-Coder-1.3B-Ins models under different combinations of outcome reward strategies.

We investigate whether combining multiple outcome reward models can further improve performance. Inspired by recent work that integrates multiple reward signals (Pourreza et al., 2025b; Ma et al., 2025), we explore various combinations among EX, GMNScore, RelPM, and AstPM. As shown in Table 11, combining EX with RelPM achieves the best overall performance, increasing the average accuracy from 41.90% (GMNScore alone) to 43.53%. The second-best result comes from EX combined with GMNScore, reaching 42.38%.

We attribute the relatively lower performance of EX & GMNScore compared to EX & RelPM to the fact that both EX and GMNScore emphasize global correctness. This may lead to redundant or even conflicting feedback, limiting the effectiveness of their combination. In contrast, the integration of RelPM provides localized partial matching signals, which offer complementary supervision and improve the performance.

Moreover, combining AstPM with either GMNScore or EX does not outperform using GMNScore or EX alone. We suspect this is due to AstPM's focus on surface-level syntax. It may penalize syntactically different yet semantically equivalent SQLs, introducing false negatives that degrade training quality. As a result, this syntactic noise may undermine the semantic robustness provided by GMNScore or EX.

While GMNScore & RelPM performs better than RelPM alone, it still falls short of GMNScore alone. We believe this is because RelPM, although based on ROT to capture richer structural semantics, still relies on partial matching algorithm, which struggles to fully capture deep semantic equivalence. By contrast, GMNScore directly leverages graph matching to assess functional equivalence, demonstrating strong robustness and stability.

It is worth noting, however, that the effectiveness of combining multiple outcome rewards is significantly lower than that of our proposed Stepwise Reward model, StepRTM. The highest value achieved by the outcome rewards combination is 43.53%, whereas StepRTM achieves a higher value of **45.32%**, representing an improvement of **1.79%**. This highlights the effectiveness of our designed stepwise reward model StepRTM. More importantly, StepRTM introduces no additional costs in terms of database execution time or GPU memory consumption.

[5]AUC is used as a metric to assess the accuracy of similarity evaluation (Zhan et al., 2025). In our experiments, a score of 100% does not indicate optimal performance, as false negatives are present in the execution outcomes.

Method	Spider	BIRD	Avg.
DeepSeek-Coder-1.3B-Ins	39.56	11.34	25.45
+ PPO w/ AstPM	51.26	11.47	31.37
+ PPO w/ RelPM	51.45	14.73	33.09
+ PPO w/ BTRM	51.35	15.45	33.40
+ PPO w/ EX	52.71	**15.84**	34.28
+ PPO w/ GMNScore (Our)	**53.87**	15.58	**34.73**

Table 12: TS Performance of DeepSeek-Coder-1.3B-Ins models under different outcome rewards with BIRD-Train.

J Generalization of GMNScore Reward

Although the GMN of GMNScore reward model is trained solely on SQL equivalence data derived from the Spider dataset, it exhibits strong cross-database generalization without requiring any additional fine-tuning. As shown in Table 12, when RL is conducted on the training split of BIRD, GMNScore continues to outperform execution-guided (EX) reward in average TS performance, consistent with the results on training split of Spider.

More broadly, we observe that across both training configurations of Spider and BIRD, our GMNScore consistently achieves higher average validation performance than EX. However, a finer-grained comparison reveals a nuanced trend: on both DeepSeek-Coder-1.3B and 6.7B with PPO, GMNScore consistently outperforms EX on the Spider evaluation dataset but shows a slight performance drop on the BIRD evaluation dataset.

We attribute this discrepancy to the inherent limitations of the GMN model itself. Although we enhanced GMN through data augmentation based on the Spider-train-pair dataset (Zhan et al., 2025), the augmented data is still constructed from Spider-style SQL, which biases the model toward better alignment with the structural and stylistic patterns of Spider. In contrast, the BIRD dataset is significantly more challenging, containing more complex SQL constructs such as deeply nested subqueries. As a result, GMN may struggle to judge equivalence in BIRD-style queries as accurately as its performance in Spider. For example, GMN is likely to score generated SQL in Spider-like styles more accurately, leading to more consistent reinforcement signals, while its scoring fidelity for BIRD-like styles is relatively weaker.

In contrast, the EX reward is purely execution-based and thus remains more robust to variations in dataset complexity and style. However, EX also suffers from false positives. Despite this slight drop in the BIRD dataset, GMNScore offers substantially larger improvements on Spider and achieves the best overall average performance. We further hypothesize that stronger base models can compensate for the small drop in reward signal quality in the BIRD dataset.

As discussed previously, GMNScore exhibits slightly reduced BIRD performance compared to EX when using Deepseek-Coder-1.3b/6.7b-Ins models, likely due to GMN's limited generalization on BIBD-style SQL. However, this performance gap decreases with stronger base models. As shown in Table 13, when using Qwen-Coder-7B and 14B as the policy backbone, GMNScore not only consistently outperforms EX on Spider, but also achieves superior results on BIRD. This further supports the effectiveness and scalability of our reward model framework across different model capacities.

K Detailed Effectiveness of GMNScore Across GRPO

Method	Spider	BIRD	Avg.
Qwen2.5-Coder-7B-Ins	62.67	22.69	42.68
+ PPO w/ EX	75.24	29.01	52.13
+ PPO w/ GMNScore (Ours)	**76.89**↑1.65	**29.60**↑0.59	**53.25**↑1.12
+ GRPO w/ EX	76.40	28.49	52.45
+ GRPO w/ GMNScore (Ours)	**78.53**↑2.13	**29.92**↑1.43	**54.23**↑1.78
Qwen2.5-Coder-14B-Ins	71.08	29.29	50.19
+ PPO w/ EX	74.66	32.98	53.82
+ PPO w/ GMNScore (Ours)	**77.37**↑2.71	**33.70**↑0.72	**55.54**↑1.72
+ GRPO w/ EX	77.56	33.90	55.73
+ GRPO w/ GMNScore (Ours)	**78.34**↑0.78	**34.35**↑0.45	**56.35**↑0.62

Table 13: TS Performance of Qwen2.5-Coder-7B/14B-Ins models directly trained by PPO/GRPO under EX and GMNScore outcome rewards.

As shown in Table 13, our proposed GMNScore consistently outperforms the EX across both PPO and GRPO training paradigms, and for both Qwen2.5-Coder-7B-Ins and 14B-Ins models. On the 7B model, GMNScore improves the average score from 52.13% to 53.25% under PPO training (**+1.12%**), and from 52.45% to 54.23% under GRPO training (**+1.78%**). Similarly, on the 14B model, GMNScore boosts the average score by **+1.72%** under PPO and **+0.62%** under GRPO. These consistent improvements demonstrate that GMNScore provides a stable reward signal than EX. Overall, these results confirm the superiority of GMNScore in guiding RL-based Text-to-SQL tasks more effectively than execution-based EX reward model.

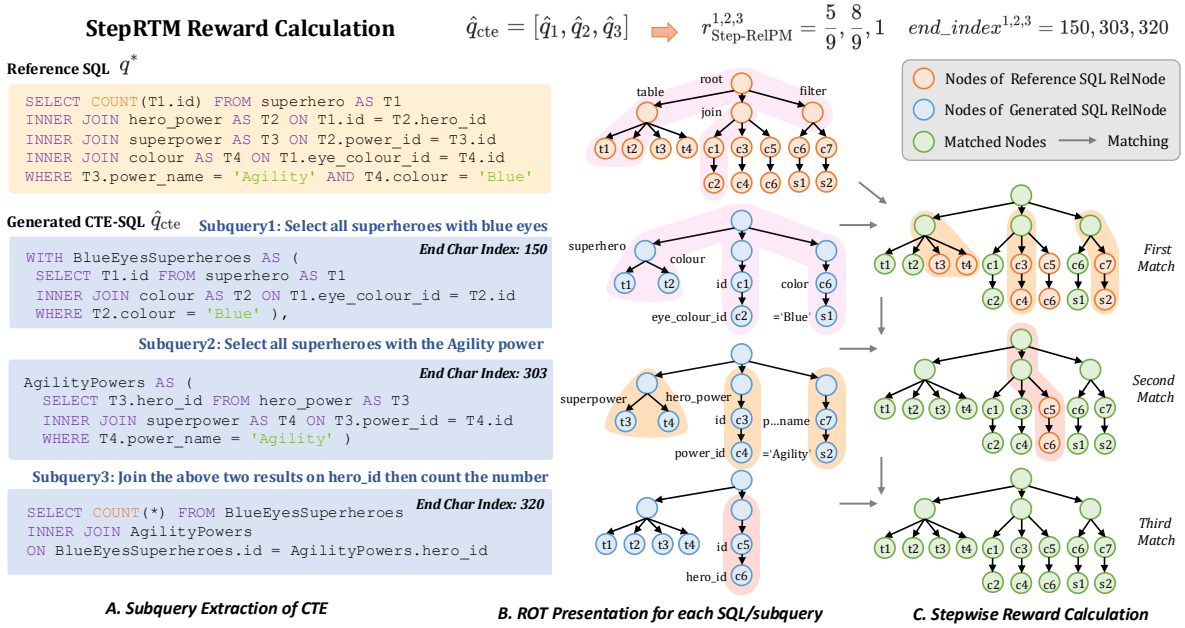

Figure 7: Detailed Overview of the StepRTM Stepwise Reward Calculation.

L StepRTM Calculation

In Figure 2 in Methodology part, we presented an extremely simplified example. As shown in Figure 7, we offer a figure with more information to help better understand the StepRTM calculation process.

M Training Details of Neural Reward Model with Language odeling

Preference Data Construction We construct preference data using the **Spider-Train Pair** from GMN's training set. Preference pairs are formed through label-driven selection, where chosen responses strictly correspond to label $= 1$, mapped from the `new_labels` column, while rejected responses (negative samples) come from the same prompt cluster. Within each group, all possible pairs between chosen (label $= 1$) and rejected (label $\neq 1$) responses are generated.

Reward Model Training To train the reward model in the above preference pairs, we fine-tune the pretrained model DeepSeek-Coder-1.3B-Instruct with a reward head. Training is carried out using an effective batch size of 16 with gradient accumulation 2. We use the AdamW optimizer, a learning rate of 1.41×10^{-5}, and a linear warm-up schedule. The model is trained for five epochs with a sequence length of 2048 tokens.

Prompt Engineering Strategy The reward model should effectively evaluates the generated SQL based on both syntactic clarity and execution-level correctness. To reinforce schema understanding and functional equivalence, the data was structured as follows:

Dataset

```
[Table Schema]
[Question]
[Reference SQL]
Please give a SQL that is functionally
identical to the SQL above:
[Chosen SQL/Rejected SQL]
```

N RelNode-based Partial Matching

To enable semantic-level comparison of SQL queries, each query is converted into a Relational Operator Tree (ROT)—a representation of the query's logical execution plan. Each ROT is constructed by parsing the SQL into relational algebra expressions that capture the sequence and dependency of logical operations (Cyganiak, 2005). In practice, this conversion is performed using Apache Calcite (Begoli et al., 2018), which yields a canonical ROT structure referred to as a RelNode. [6] The RelNode abstraction refines the logical plan via operator reordering and redundant clause elimination, making it robust to syntactic variations while preserving execution semantics.

Based on RelNode representations, a partial matching strategy is employed to measure the sim-

[6] https://github.com/apache/calcite.

ilarity between the relational operator trees of the reference SQL $R(q^\star)$ and the generated SQL $R(\hat{q})$. Let $\mathcal{N}_{q^\star}$ and $\mathcal{N}_{\hat{q}}$ denote the sets of nodes in the RelNodes of the reference and generated queries, respectively. A node pair (n, n') is considered a match, denoted as match(n, n'), if the two nodes share the same relational operator type and their associated attributes are semantically equivalent.

To determine the best match for a node $n' \in \mathcal{N}_{\hat{q}}$, we score it against all candidate nodes from $\mathcal{N}_{q^\star}$, selecting the highest-scoring candidate. Each candidate score m^j is computed recursively as a weighted sum of node-level similarity and the average similarity of their child nodes:

$$m^j = \alpha \cdot m^j_{\text{self}} + (1-\alpha) \cdot \frac{1}{N} \sum_{i=1}^{N} m^j_{\text{child}^{(i)}}$$

where $\alpha \in (0, 1)$, and a smaller value of α emphasizes structural alignment by favoring subtree similarity, while a larger value prioritizes node-level matching accuracy.

Based on the above definition, the precision and recall of the matching results are computed as:

$$\text{Precision} = \frac{|\{n \in \mathcal{N}_{\hat{q}} \mid \exists n' \in \mathcal{N}_{q^\star},\ \text{match}(n, n')\}|}{|\mathcal{N}_{\hat{q}}|}$$

$$\text{Recall} = \frac{|\{n \in \mathcal{N}_{q^\star} \mid \exists n' \in \mathcal{N}_{\hat{q}},\ \text{match}(n, n')\}|}{|\mathcal{N}_{q^\star}|}$$

Here, match(n, n') denotes a binary function that returns 1 if nodes n and n' are matched, and 0 otherwise. In the context of SQL generation, capturing the semantics of the reference SQL is prioritized. The final reward r_{RelPM} is calculated using a relatively large β to emphasize recall as follows:

$$r_{\text{RelPM}} = \frac{(1+\beta^2) \cdot \text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}} \quad (1)$$

O FuncEvalGMN

RelNode presentations are firstly convert into graphs. Nodes in the graphs represent relational operators or expressions, while edges capture both logical execution dependencies and data flow relations. Nodes represent relational operators or expressions, and edges capture execution dependencies and data flows. A GMN (Li et al., 2019) encodes these graphs through three stages: inner-graph message passing, cross-graph semantic alignment, and gated aggregation.

In the inner-graph message passing stage, node embeddings are iteratively updated by aggregating information from their local neighborhoods:

$$m_v^{(t+1)} = \sum u \in N(v) f_{\text{inner}}(h_v^{(t)}, h_u^{(t)}, e_{uv}), \quad (2)$$

where $h_v^{(t)}$ and $h_u^{(t)}$ denote the hidden representations of nodes v and u at step t, and e_{uv} is the edge embedding derived from a learned layer.

Next, the cross-graph message passing stage aligns structurally similar components between $\mathcal{G}\hat{q}$ and $\mathcal{G}q^*$ using cross-attention:

$$r_v^{(t)} = \text{MLP}(h_v^{(t)} \oplus p_v^{(t)}),$$
$$\mu_v^{(t+1)} = \sum_{u \in \mathcal{G}_2(v)} a_{u \to v}(r_v^{(t)} - r_u^{(t)}), \quad (3)$$

where $p_v^{(t)}$ is the positional encoding and $a_{u \to v}$ is the attention weight defined by:

$$a_{u \to v} = \frac{\exp(s(r_v^{(t)}, r_u^{(t)}))}{\sum_{u' \in \mathcal{G}_2(v)} \exp(s(r_v^{(t)}, r_{u'}^{(t)}))}, \quad (4)$$

where $s(r_v, r_u) = \frac{r_v \cdot r_u}{\sqrt{d}}$ and d is the dimensionality of node embeddings. This cross-graph attention mechanism enables fine-grained alignment between substructures in different SQL queries. After T propagation steps, the graph-level representation is computed via a gated aggregation mechanism (Li et al., 2015), which selectively emphasizes salient node features:

$$h_G = \text{MLP}_G\left(\sum_{v \in V} \sigma\left(\text{MLP}_{\text{gate}}(h_v^{(T)})\right) \odot \text{MLP}(h_v^{(T)})\right), \quad (5)$$

where $\sigma(\cdot)$ is the sigmoid function and $\odot$ denotes element-wise multiplication. To quantify semantic similarity between $\hat{q}$ and $q^\star$, we compute the negative Euclidean distance between their final graph embeddings:

$$r_{\text{similarity}} = -\left\| h_{G_{\hat{q}}} - h_{G_{q^*}} \right\|_2 \quad (6)$$

P Prompt Strategy of CTE

We incorporate CTE-style data into the supervised fine-tuning (SFT) phase to encourage the generation of Common Table Expressions (CTEs). Specifically, we augment the training set by rewriting examples from the BIRD-train dataset. Each example is first passed through two prompting stages using GPT-4o: (1) determining whether the original SQL query is complex enough to benefit from CTE rewriting, and (2) if so, performing the actual CTE transformation. We then evaluate the functional equivalence of the rewritten and original SQL queries using Test Suite Accuracy to ensure correctness. This pipeline yields 3,252 rewritten examples (BIRD-train-CTE) and 3,810 unmodified examples that were deemed unnecessary to rewrite. The prompts and an example transformation are shown below.

P.1 CTE Rewriting Necessity Judgment

Instruction

You are provided with the following SQL statement:
{Reference_SQL}
Evaluate the complexity of the given SQL query.
Determine if splitting it into Common Table Expressions (CTEs) using WITH x AS clauses would help with understanding. If the query is complex and CTEs would be beneficial, output {{"cte_necessary": "True"}}.
If the query is simple and does not require CTEs (for example, a basic single - table SELECT with simple conditions like SELECT SUM(occurrences) FROM words WHERE LENGTH(word) = 3), output {{"cte_necessary": "False"}}.

P.2 CTE Rewriting

Instruction

Text-to-SQL is a task of transforming natural language queries into Structured Query Language (SQL) statements.
Table Schema:
{prompt_schema}
Question:
{query}
You are given the following SQL statement as an response (GroundTruth_SQL):
{Reference_SQL}
Rewrite above SQL query into a Common Table Expression (CTE) format using 'WITH x AS' clauses. Break the query into logical steps and use intermediate CTEs for each step.
The response format should be:
{{"sql": "<CTE-formatted SQL statement>"}}

P.3 CTE Rewriting Example

Type	Content
Question	What is the shipment ID of the heaviest shipment that Zachery Hicks transported?
Reference SQL	SELECT T1.ship_id FROM shipment AS T1 INNER JOIN driver AS T2 ON T1.driver_id = T2.driver_id WHERE T2.first_name = 'Zachery' AND T2.last_name = 'Hicks' ORDER BY T1.weight DESC LIMIT 1
Reference CTE SQL	WITH DriverShipments AS (SELECT T1.ship_id, T1.weight FROM shipment AS T1 INNER JOIN driver AS T2 ON T1.driver_id = T2.driver_id WHERE T2.first_name = 'Zachery' AND T2.last_name = 'Hicks'), HeaviestShipment AS (SELECT ship_id FROM DriverShipments ORDER BY weight DESC LIMIT 1) SELECT ship_id FROM HeaviestShipment

Q Prompt Strategy of Text-to-SQL Task

Instruction

There are two tables: singer, song.
the structure of table singer is as follows:
column name	column type
————	————
singer_id	number
name	text
birth_year	number
net_worth_millions	number
citizenship	text
singer_id is the primary key.

the structure of table song is as follows:
column name	column type
————	————
song_id	number
title	text
singer_id	number
sales	number
highest_position	number
song_id is the primary key.
The singer_id is the foreign key, reference to singer_id in table singer.

Question: What is the sname of every sing that does not have any song?
The corresponding SQL code is:

Reference SQL (Ground Truth SQL)

SELECT Name FROM singer WHERE Singer_ID NOT IN (SELECT Singer_ID FROM song)

Example Generated SQL

SELECT name FROM singer WHERE NOT singer_id IN (SELECT singer_id FROM song)

R Reasoning Prompt Strategy of Text-to-SQL Task (for Qwen2.5-Coder-7B)

Instruction with Reasoning

You are an experienced database expert. Given the database schema and the user's question, your task is to write a single SQL that correctly answers the question.

Database schema:
There are two tables: manufacturers, products.
the structure of table manufacturers is as follows:
column name	column type
————	————
code	number
name	text
.....
the structure of table products is as follows:
.....
Question: What is the average price across all products?

Respond in the following format:
<think>
Your detailed and step-by-step thinking path toward finding the correct SQL query
</think>
<answer>
“‘sql
Your predicted SQL query
“‘
</answer>
Now is your turn to respond in the above format.

Example Generated Response

<think>
1. **Understand the Task**: The task is to find the total hours of all projects in the database.
2. **Identify Relevant Table**: From the schema, the table that contains the project hours is the ‘projects‘ table.
3. **Identify Relevant Column**: The column in the ‘projects‘ table that contains the hours is ‘hours‘.
4. **Formulate the SQL Query**: To get the total hours, we need to sum up all the values in the ‘hours‘ column of the ‘projects‘ table.
</think>
<answer>
“‘sql
SELECT headquarters
FROM companies
GROUP BY headquarters
HAVING COUNT(*) > 2;
“‘
</answer>

S Case Study: Outcome Reward Comparison

To illustrate the behavioral differences of reward models under the outcome-based reward setting, we present four case studies comparing responses generated by models trained with SFT, BTRM, EX and GMN rewards.

Table 14: Sample query, responses from the model trained with Reward model based GMN, BTRM and Execution. If it is an obvious error, we color the text in the error location red.

Type	Content	TS
Prompt	There are four tables: city, sqlite_sequence, country, countrylanguage. the structure of table city is as follows: ... the structure of table sqlite_sequence is as follows: ... the structure of table country is as follows: | column name | column type | | ———— | ———— | | code | text | | name | text | | continent | text | | region | text | | surfacearea | number | | indepyear | number | | population | number | | lifeexpectancy | number | | gnp | number | | gnpold | number | | localname | text | | governmentform | text | | headofstate | text | | capital | number | | code2 | text | code is the primary key. the structure of table countrylanguage is as follows: ... Question: Find the government form name and total population for each government form whose average life expectancy is longer than 72.	N/A
Groundtruth	SELECT sum(Population) , GovernmentForm FROM country GROUP BY GovernmentForm HAVING avg(LifeExpectancy) > 72	N/A
SFT	select governmentform, sum(population) from country where lifeexpectancy > 72 group by governmentform	0
BTRM	SELECT governmentform, SUM(population) FROM country WHERE lifeexpectancy > 72 GROUP BY governmentform	0
EX	SELECT governmentform, SUM(population) FROM country WHERE lifeexpectancy > 72 GROUP BY governmentform	0
GMN	SELECT governmentform, SUM(population) FROM country GROUP BY governmentform HAVING AVG(lifeexpectancy) > 72	1

Table 15: Sample query, responses from the model trained with Reward model based GMN, BTRM and Execution. If it is an obvious error, we color the text in the error location red.

Type	Content	TS
Prompt	There are six tables: continents, countries, car_makers, model_list, car_names, cars_data. the structure of table continents is as follows: ... the structure of table countries is as follows: ... the structure of table car_makers is as follows: ... the structure of table model_list is as follows: ... the structure of table car_names is as follows: | column name | column type | | ———— | ———— | | makeid | number | | model | text | | make | text | makeid is the primary key. The model is the foreign key, reference to model in table model_list. the structure of table cars_data is as follows: | column name | column type | | ———— | ———— | | id | number | | mpg | text | | cylinders | number | | edispl | number | | horsepower | text | | weight | number | | accelerate | number | | year | number | id is the primary key. The id is the foreign key, reference to makeid in table car_names. Question: What is the accelerate of the car make amc hornet sportabout (sw)?	N/A
Groundtruth	SELECT T1.Accelerate FROM CARS_DATA AS T1 JOIN CAR_NAMES AS T2 ON T1.Id = T2.MakeId WHERE T2.Make = 'amc hornet sportabout (sw)';	N/A
SFT	select t2.accelerate from car_makers as t1 join cars_data as t2 on t1.id = t2.id where t1.make = "amc" and t1.model = "hornet sportabout"	0
BTRM	SELECT cars_data.accelerate FROM cars_data INNER JOIN car_names ON cars_data.makeid = car_names.makeid WHERE car_names.model = 'amc hornet sportabout (sw)'	0
EX	SELECT accelerate FROM cars_data WHERE id IN (SELECT makeid FROM car_names WHERE model = 'sportabout (sw)')	0
GMN	SELECT cars_data.accelerate FROM cars_data INNER JOIN car_names ON cars_data.id = car_names.makeid WHERE car_names.make = 'amc hornet sportabout (sw)'	1

Table 16: Sample query, responses from the model trained with Reward model based GMN, BTRM and Execution. If it is an obvious error, we color the text in the error location red.
GT Explanation: from the CAR_NAMES table and the CARS_DATA table by associating the MakeId with the Id, sort the rows by mpg descending order and take the first record.

Type	Content	TS
Prompt	There are six tables: continents, countries, car_makers, model_list, car_names, cars_data. ... the structure of table model_list is as follows: | column name | column type | | ———— | ———— | | modelid | number | | maker | number | | model | text | modelid is the primary key. The maker is the foreign key, reference to id in table car_makers. the structure of table car_names is as follows: | column name | column type | | ———— | ———— | | makeid | number | | model | text | | make | text | makeid is the primary key. The model is the foreign key, reference to model in table model_list. the structure of table cars_data is as follows: | column name | column type | | ———— | ———— | | id | number | | mpg | text | | cylinders | number | | edispl | number | | horsepower | text | | weight | number | | accelerate | number | | year | number | id is the primary key. The id is the foreign key, reference to makeid in table car_names. Question: Which model saves the most gasoline? That is to say, have the maximum miles per gallon.	N/A
Groundtruth	SELECT T1.Model FROM CAR_NAMES AS T1 JOIN CARS_DATA AS T2 ON T1.MakeId = T2.Id ORDER BY T2.mpg DESC LIMIT 1;	N/A
SFT	select t2.model from cars_data as t1 join model_list as t2 on t1.id = t2.modelid group by t1.id order by t1.edispl desc limit 1	0
BTRM	SELECT model FROM cars_data ORDER BY mpg DESC LIMIT 1	0
EX	SELECT model FROM model_list ORDER BY mpg DESC LIMIT 1	0
GMN	SELECT car_names.model FROM cars_data INNER JOIN car_names ON cars_data.id = car_names.makeid ORDER BY cars_data.mpg DESC LIMIT 1	1

Table 17: Sample query, responses from the model trained with Reward model based GMN, BTRM and Execution. If it is an obvious error, we color the text in the error location red.
Error Explanation of SFT: The id equivalents of tv_channel and cartoon have different meanings and should not be connected simply by using the id.
Error Explanation of BTRM: Question focused on cartoon is not suitable to use Inner join channel table, no foreign key connection, should not assume that all channels in the cartoon table in the database have a corresponding id stored in tv_channel.

Type	Content	TS
Prompt	There are three tables: tv_channel, tv_series, cartoon. the structure of table tv_channel is as follows: | column name | column type | | ———— | ———— | | id | text | | series_name | text | | country | text | | language | text | | content | text | | pixel_aspect_ratio_par | text | | hight_definition_tv | text | | pay_per_view_ppv | text | | package_option | text | id is the primary key. the structure of table tv_series is as follows: | column name | column type | ... the structure of table cartoon is as follows: | column name | column type | | ———— | ———— | | id | number | | title | text | | directed_by | text | | written_by | text | | original_air_date | text | | production_code | number | | channel | text | id is the primary key. Question: find the package option of the tv channel that do not have any cartoon directed by Ben Jones.	N/A
Groundtruth	SELECT package_option FROM TV_Channel WHERE id NOT IN (SELECT channel FROM cartoon WHERE directed_by = 'Ben Jones')	N/A
SFT	select distinct package_option from tv_channel except select t1.package_option from tv_channel as t1 join cartoon as t2 on t1.id = t2.id where t2.directed_by = "Ben Jones"	0
BTRM	SELECT tv_channel.package_option FROM tv_channel EXCEPT SELECT tv_channel.package_option FROM tv_channel INNER JOIN cartoon ON tv_channel.id = cartoon.channel WHERE cartoon.directed_by = "Ben Jones"	0
EX	SELECT package_option FROM tv_channel WHERE NOT id IN (SELECT directed_by FROM cartoon WHERE directed_by = 'Ben Jones')	0
GMN	SELECT package_option FROM tv_channel WHERE NOT id IN (SELECT channel FROM cartoon WHERE directed_by = 'Ben Jones')	1