Mixed-Policy GRPO for Text-to-SQL with Off-Policy Data Generation

Marko Sterbentz[♠,♣] **Michael Robert Glass**[♣] **Nhan Huu Pham**[♣]
Dharmashankar Subramanian[♣] **Kristian J. Hammond**[♠]

[♠]Northwestern University [♣]IBM Research

Abstract

Recent advances in text-to-SQL have shown that methods such as Group Relative Policy Optimization (GRPO) can substantially improve reasoning performance, but these approaches remain inherently on-policy, limiting their ability to incorporate novel reasoning patterns. In this work, we address this limitation by leveraging existing datasets to generate high-quality off-policy rollouts, enabling mixed-policy training that exposes models to diverse and informative reasoning trajectories. We present the first application of mixed-policy GRPO to the text-to-SQL domain and introduce a systematic study of off-policy data generation strategies for this setting, including a novel method, Iterative Error Correction (IEC), which iteratively refines model outputs through targeted feedback. Our experiments show that mixed-policy GRPO outperforms both base models and on-policy GRPO, yielding average improvements of +4.7% over base models and +4.1% over on-policy GRPO across the Spider and BIRD benchmarks. Gains are particularly strong on BIRD, reaching up to +7.3% over base models and +4.5% over on-policy GRPO.

1 Introduction

The text-to-SQL task aims to translate natural language questions into executable SQL queries that retrieve correct answers from a given database (Zelle and Mooney, 1996). A central challenge lies in mapping the linguistic variability of natural language to the precise, schema-dependent logical forms required by SQL. Recent progress in text-to-SQL with large language models (LLMs) has been driven by two key factors: access to larger training datasets and advances in training algorithms. To address data scarcity, many approaches generate synthetic question–query pairs to augment supervised training data (Weir et al., 2020; Yu et al., 2020; Yang et al., 2024; Li et al., 2025).

In parallel, Group Relative Policy Optimization (GRPO) (Shao et al., 2024) has emerged as an effective method for improving the reasoning capabilities of LLMs on complex tasks, and it has been successfully applied to text-to-SQL (Ma et al., 2025; Pourreza et al., 2025; Yao et al., 2026). Despite these successes, existing GRPO-based approaches face two key limitations. First, they are inherently on-policy: models are trained only on rollouts sampled from their current policy. Prior work shows that such on-policy reinforcement learning (RL) tends to amplify existing reasoning patterns rather than encouraging the discovery of new ones (Zhao et al., 2025; Yue et al., 2025). Second, GRPO is more computationally expensive than supervised fine-tuning, making it costly to scale.

Mixed-policy GRPO methods have been proposed to address these limitations by incorporating off-policy rollouts during training, enabling more effective use of available data without increasing dataset size. These off-policy rollouts are produced by a separate model or process. A central challenge in this setting is the generation of high-quality off-policy reasoning traces. For example, (Nath et al., 2025) introduces prompt-level hints to induce off-policy rollouts, while LUFFY (Yan et al., 2025) relies on an auxiliary model to generate off-policy data prior to training. For mathematical reasoning, these approaches consistently outperform purely on-policy methods, suggesting that exposure to diverse reasoning trajectories is beneficial.

In this work, we present the first application of mixed-policy GRPO to the text-to-SQL domain, adapting prior formulations to accommodate structured query generation and execution-based evaluation. We systematically investigate multiple off-policy data generation strategies within this framework, including a novel method, Iterative Error Correction (IEC), which incrementally refines model outputs through targeted feedback. Together, these contributions enable mixed-policy training that ex-

Proceedings of the First Workshop on Structured Understanding, Retrieval, and Generation in the LLM Era (SURGeLLM 2026), pages 313–325

poses models to novel reasoning patterns and improves text-to-SQL reasoning performance. Our results show that the strongest mixed-policy GRPO configuration outperforms both base models and on-policy GRPO, yielding average improvements of +4.73% over base models and +4.11% over on-policy GRPO across the Spider and BIRD benchmarks. We observe particularly strong gains on BIRD, where the best-performing configuration achieves improvements of +7.3% over base models and +4.51% over on-policy GRPO.

We find that the the effectiveness of off-policy data is dependent on the training strategy, suggesting that data generation and policy optimization are tightly coupled. In particular, the utility of different off-policy generation varies across mixed-policy settings. We find that IEC is especially effective in simpler mixed-policy settings, where it produces higher-quality supervision and leads to stronger downstream performance.

Our contributions can be summarized as follows:

- We present the first application of mixed-policy GRPO to text-to-SQL, adapting it to structured query generation with execution-based evaluation.
- We conduct a systematic study of off-policy data generation strategies for mixed-policy GRPO, including the introduction of Iterative Error Correction (IEC).
- We show that mixed-policy GRPO outperforms both base models and on-policy GRPO, achieving gains of up to +7.3% on BIRD and improving performance across both Spider and BIRD benchmarks.
- We show that IEC is particularly effective in simpler mixed-policy regimes, providing higher-quality supervision and improved downstream performance.

2 Related Work

Reinforcement Learning with Verifiable Rewards (RLVR) RLVR has emerged as a powerful paradigm for improving chain-of-thought reasoning in language models (Shao et al., 2024; Liu et al., 2025). By defining a rule-based reward function, model outputs can be evaluated and compared, allowing the model to learn to imitate higher-scoring reasoning traces while avoiding lower-scoring ones. This approach has been successfully applied across a variety of domains, including code generation (Du et al., 2025), medical question answering (Zhang et al., 2025a,b; Adly et al., 2025), and text-to-SQL (Ma et al., 2025; Pourreza et al., 2025; Yao et al., 2026). More recent work extends this paradigm through mixed-policy or off-policy training, which incorporates externally sourced reasoning traces to expose models to new patterns and further boost performance (Yan et al., 2025; Nath et al., 2025). However, these mixed-policy approaches have so far focused primarily on mathematical reasoning and have not been explored in the text-to-SQL domain.

Synthetic Text-to-SQL Data Generation Prior work addresses data scarcity in text-to-SQL by generating synthetic question–SQL pairs using template- or grammar-based methods (Guo et al., 2018; Weir et al., 2020; Yu et al., 2020; Sterbentz et al., 2026), question generation for data augmentation (Zhong et al., 2020; Wang et al., 2021; Wu et al., 2021; Guo et al., 2025), and LLM-based approaches (Yang et al., 2024; Li et al., 2025). SynSQL (Li et al., 2025) further augments such data with chain-of-thought traces by conditioning on the target SQL query. However, providing the ground-truth query can lead to shallow, procedural reasoning rather than detailed schema- and join-level understanding. In contrast, Iterative Error Correction (IEC) withholds the target query and instead provides intermediate hints, encouraging more fine-grained reasoning traces.

Error Correction and Prompt Optimization Error correction has proven effective for improving LLM outputs, with prior work refining predictions at inference time by diagnosing errors and providing targeted feedback (Yan et al., 2023; Shinn et al., 2023; Madaan et al., 2023). Iterative Error Correction (IEC) follows a similar paradigm but applies it to training data generation rather than inference. Related approaches incorporate rationales or reasoning traces into training (Zelikman et al., 2024; He et al., 2025), often relying on pre-defined samples, whereas IEC generates and refines them dynamically. Other methods introduce hints during reasoning-focused training (Nath et al., 2025), but do not verify their correctness. In contrast, IEC iteratively updates hints until they provide reliable guidance. Finally, prompt optimization methods refine instructions to improve inference-time behavior (Pryzant et al., 2023; Gao et al., 2025; Agrawal

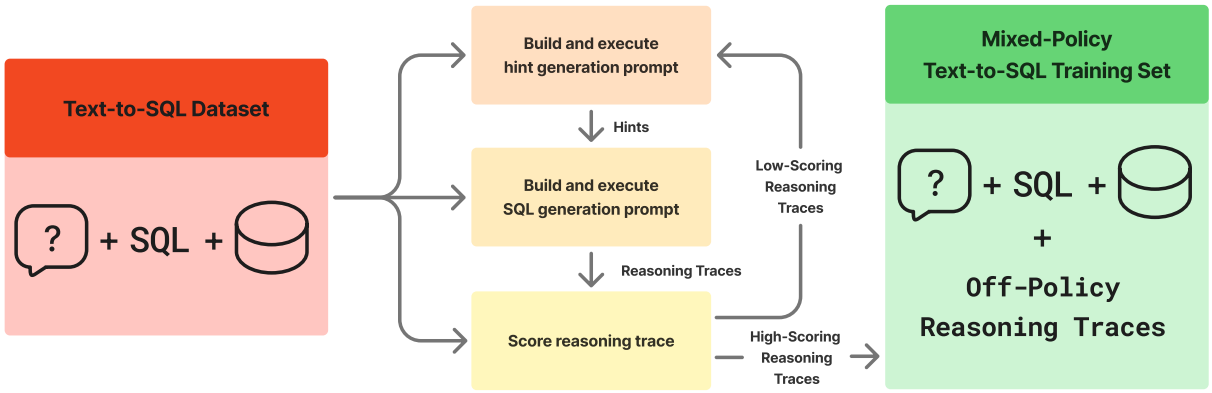

Figure 1: The full Iterative Error Correction process, which takes in a text-to-SQL dataset consisting of questions, SQL queries, and databases and produces high-quality off-policy reasoning traces for each example in the dataset.

et al., 2026), whereas IEC focuses on producing high-quality reasoning traces for training.

3 Iterative Error Correction for Off-Policy Data Generation

To enable mixed-policy training, we require a source of high-quality off-policy reasoning traces for text-to-SQL. The typical process for generating these begins with a set of question–SQL pairs. A straightforward approach is to prompt an off-the-shelf model to generate one or more candidate traces and then select the best sample according to a scoring function (Yan et al., 2025). Each reasoning trace consists of the LLM's output, including both the final SQL query and the intermediate chain-of-thought used to derive it. However, this approach often yields incorrect or inconsistent outputs for complex tasks such as text-to-SQL, even when using large models.

To address this limitation, we apply error-correction techniques that identify flaws in the generated SQL and provide this feedback to the model, prompting it to regenerate the query while explicitly avoiding previously identified errors. This procedure is repeated iteratively until either a correct SQL query is produced or a predefined maximum number of iterations is reached. We refer to this process as **Iterative Error Correction (IEC)**. An overview of the full pipeline is shown in Figure 1.

At a high level, data generation with IEC proceeds in three primary steps for each question–SQL pair from an existing text-to-SQL dataset. First, a model is prompted to reason about potential errors that could arise when generating the target SQL and to produce a set of hints aimed at avoiding such errors. Second, these hints are incorporated into a SQL generation prompt, which is used to produce an off-policy reasoning trace consisting of both the reasoning process and the resulting SQL query. Third, the generated output is scored. If the output is erroneous or receives a low score, the process is restarted with an updated set of hints derived from the previously generated incorrect SQL.

It is important to note that the ground-truth SQL is not directly incorporated into the SQL generation prompt. Providing the target query explicitly can cause the model to rigidly adhere to the ground truth, effectively short-circuiting the reasoning process required to derive the correct SQL. This typically results in lower-quality reasoning traces that describe the solution at a high level rather than reasoning explicitly about the database schema and SQL syntax. Instead, we use the ground-truth SQL only in a separate hint-generation step. As a result, when generating the target SQL, the model never has direct access to the ground-truth query, but instead relies solely on indirect guidance of the generated hints.

3.1 Generating Hints to Fix Errors

The first step in producing off-policy reasoning traces is to generate a set of informative hints to guide the model in reasoning about how to generate the target SQL. During hint generation, the model is instructed to reason about potential errors that may arise when converting a natural language question into SQL. The model is provided with the target question, the database schema, and the ground-truth SQL. It is prompted to use chain-of-thought reasoning to identify likely failure modes, enclosing its reasoning within `<think>...</think>` tags and the resulting set of hints within `<hint>...</hint>` tags. The full prompt used for the initial iteration is shown in

Figure 3 in Appendix A.

In successive iterations, the model is provided with the set of hints generated in previous iterations as well as the erroneous SQL output from the most recent iteration. The model is instructed to compare the predicted SQL against the ground-truth SQL, identify new errors, and update the hint set accordingly. Rather than simply appending new hints, the model is explicitly instructed to revise the existing set, allowing outdated or incorrect hints to be removed. This prevents the accumulation of stale guidance that could otherwise mislead the model in later iterations. This iterative hint refinement continues until a correct SQL query is produced or a predefined maximum number of IEC iterations is reached. The prompt used for successive iterations is shown in Figure 4 in Appendix A.

Using these prompts, an off-the-shelf LLM produces a set of error-aware hints based on discrepancies it identifies between the predicted and ground-truth SQL. The hints are then extracted from the model output and used as part of the input for generating off-policy reasoning traces during SQL generation. Figure 2 provides example outputs illustrating the types of hints produced by this process.

Figure 2 illustrates a representative example of iterative hint refinement. In Iteration 0, six hints are generated using only the target question, database schema, and ground-truth SQL. Since these hints do not lead to a correct SQL query, the hint set is updated based on errors detected in the model's predicted SQL, yielding two new hints in Iteration 1. After another unsuccessful attempt, the hints are further revised in Iteration 2, resulting in additional refinements to the existing hints. This final set of hints successfully guides the model to produce the correct SQL query, and the associated reasoning trace is retained for use in mixed-policy training.

3.2 Generating Text-to-SQL Reasoning

To generate off-policy reasoning traces for SQL production, we use an adapted version of the SQL-R1 prompt (Ma et al., 2025) that incorporates the error hints generated by IEC. The prompt specifies the overall task and required output structure, and includes the database schema, the current set of error-aware hints, a complete example reasoning trace, and the target question. The model is instructed to enclose its reasoning within `<think>...</think>` tags and the final SQL within `<answer>...</answer>` tags. The full prompt template is shown in Figure 5 in Appendix A. Using this prompt, an off-the-shelf LLM is then used to produce a reasoning trace consisting of both the intermediate reasoning steps and final SQL query. This SQL generation step is executed at each iteration using the current set of hints.

3.3 Scoring Off-Policy Reasoning Traces

Each generated reasoning trace is assigned a scalar score using a predefined scoring function. Outputs that receive low scores are re-entered into the Iterative Error Correction loop. To be useful for training, a high-quality off-policy reasoning trace must satisfy two key criteria: (1) it must follow a structured output format that clearly delineates the reasoning process and the final SQL query, and (2) it must produce a correct SQL query.

To enforce these properties, we adopt the progressive scoring function proposed by (Ma et al., 2025). This function assigns rewards at multiple stages of evaluation. First, a format reward encourages outputs that adhere to a predefined structure, with reasoning enclosed within `<think>...</think>` tags, the answer enclosed within `<answer>...</answer>` tags, and the final SQL query formatted using ```` ```sql...``` ```` markdown within the answer. The format reward is defined as:

$$S_f = \begin{cases} 1, & \text{if the format is correct} \\ -1, & \text{if the format is incorrect} \end{cases} \quad (1)$$

Next, an execution reward ensures that the generated SQL query is executable and completes within a predefined time limit. This ensures that the query which is generated is syntactically correct. This execution reward is defined as:

$$S_e = \begin{cases} 2, & \text{if SQL is executable} \\ 0, & \text{if the format is incorrect} \\ -2, & \text{if SQL is not executable} \end{cases} \quad (2)$$

An accuracy reward further evaluates whether the executable SQL query produces the correct result for the given example:

$$S_r = \begin{cases} 3, & \text{if SQL result is correct} \\ 0, & \text{if format is incorrect} \\ & \text{or SQL is not executable} \\ -3, & \text{if SQL result is incorrect} \end{cases} \quad (3)$$

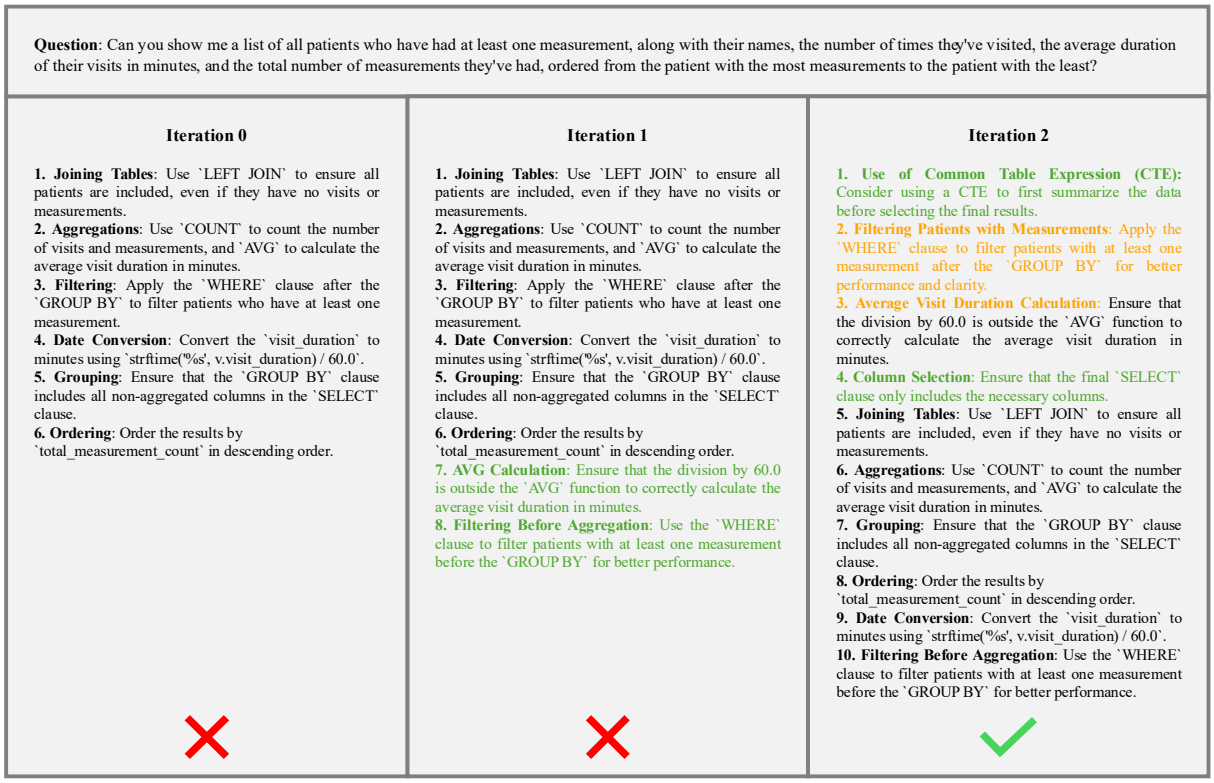

Figure 2: Hints generated during the Iterative Error Correction process. For Iteration 0, only the gold query was used for hint generation. At successive iterations, the hint generation process has access to the set of hints and the erroneously generated SQL from the previous iteration. Newly added hints are shown in green, while modified hints are shown in orange.

Finally, a length reward encourages concise reasoning while preserving correctness. This reward is applied only when the query result is correct and is defined as:

$$S_l = \begin{cases} 0.5S_{tl} + S_{al}, & \text{if correct and } l_r \leq \texttt{MAX} \\ 0.5 + S_{al}, & \text{if correct and } l_r > \texttt{MAX} \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where l_r is the total length of the response, MAX is the maximum response length desired, $S_{tl} = (len_{think} + len_{answer})$ / MAX, and $S_{al} = len_{sql}$ / len_{answer}.

The final score assigned to a reasoning trace is computed as the sum of these four reward components. For each question–SQL pair, we retain the highest-scoring reasoning trace if its score exceeds 6; otherwise, it is discarded and further IEC iterations are performed. This threshold corresponds to the minimum score required for both correct format and correct SQL execution.

4 Mixed-Policy GRPO Training for Text-to-SQL

To train a model for the text-to-SQL task, we propose a modified version of Group Relative Policy Optimization (GRPO) in which a single off-policy rollout is incorporated alongside the on-policy rollouts produced by the model during training. This formulation allows for off-policy rollouts from any source to be used.

4.1 Group Relative Policy Optimization

GRPO is a reinforcement learning technique that enables a model to improve its outputs by generating multiple candidate responses, scoring each output, and comparing them to determine relative advantages. These relative advantages are then used to update the model's policy. Because all outputs are generated according to the current policy, GRPO is inherently an on-policy method. To prevent destabilizing updates, a KL divergence term is typically added to the loss function, which discourages large changes to the policy.

A key advantage of GRPO is that outputs are evaluated using verifiable, rule-based rewards

rather than a learned reward model. This allows simple, deterministic reward functions to guide policy updates while maintaining stable learning.

4.2 Mixed-Policy GRPO

While GRPO is effective at improving model outputs through on-policy updates, it is limited to reinforcing reasoning patterns already present in the model. To accelerate learning and incorporate high-quality reasoning strategies not yet captured by the policy, we extend GRPO to a mixed-policy setting, in which a single off-policy rollout is incorporated alongside the on-policy rollouts. This modification allows the model to learn from both its own policy and external high-quality traces.

To train the model to generate SQL for a given question, we adopt a binary reward function based on SQL execution accuracy that rewards the model for producing SQL that correctly answers the target question. Formally, for a model output τ the reward function R is defined as,

$$R(\tau) = \begin{cases} 1 & \text{if the SQL query in } \tau \\ & \text{produces the correct output} \\ 0 & \text{otherwise} \end{cases} \quad (5)$$

For the training algorithm, we adopt the mixed-policy framework proposed by (Yan et al., 2025), with two key modifications. First, we omit the KL-loss term, which is normally used to regularize updates and prevent overly large changes to the model's distribution. Since our goal is to modify the core behavior of the model, we allow larger gradient updates from off-policy traces. Second, we omit policy shaping via regularized importance sampling, which re-weights the gradient of off-policy outputs to increase learning for low-probability tokens. A complete formalization can be seen in Appendix B.

5 Experimental Setup

5.1 Off-Policy Data Generation Experiments

Data Generation Methods. We compare three off-policy data generation methods that do not directly incorporate the target SQL query into the prompt in order to see which ones can produce the most useful off-policy reasoning rollouts. The first, **Direct Sampling**, prompts a model to directly generate candidate reasoning traces, from which the highest-scoring one is selected according to the same scoring function used for Iterative Error Correction. This approach does not involve any hint generation or iterative refinement. The second method, **Hint Augmented Sampling**, builds off the Direct Sampling method and incorporates an initial hint generation step, as in IEC, and prompts the model to produce candidate traces. The highest-scoring trace is then selected. However, unlike IEC, the set of hints is not updated. The third method is **Iterative Error Correction** described in Section 3. IEC builds upon the Hint Augmented Sampling method by generating hints and iteratively refining them based on errors detected in previous outputs.

To ensure a fair comparison and control for improvements arising from generating multiple outputs, all three methods produce a total of ten samples per question. We use Qwen2.5-72B-Instruct as the base model for all data generation methods.

Testing Data and Evaluation. We use the 4,000 "complex" SynSQL samples from (Ma et al., 2025) as the basis for generating off-policy rollouts for each method. Performance is measured as the percentage of examples whose generated traces satisfy both format constraints and SQL correctness under the IEC scoring function.

5.2 Mixed-Policy Training Experiments

Baselines and Models. We compare three training strategies: standard on-policy GRPO (**On-Policy**) and two mixed-policy approaches, the simple mixed-policy method proposed in Section 4.2 (**Mixed-Policy**) and **LUFFY**. On-Policy is standard GRPO and uses only rollouts sampled from the current policy during training. In contrast, both Mixed-Policy and LUFFY incorporate off-policy rollouts, with LUFFY additionally applying policy shaping mechanisms (Yan et al., 2025). Across all settings, we use Qwen2.5-Coder-3B-Instruct as the base model and train using the execution-based reward function defined in Equation 5.

Evaluation Benchmarks. We evaluate each training method on two widely used text-to-SQL benchmarks: Spider (Yu et al., 2018) and BIRD (Li et al., 2023). Spider Dev consists of 1,034 question-SQL pairs across 20 databases, while Spider Test consists of 2,147 question-SQL pairs across 40 databases. BIRD Dev consists of 1,534 question-SQL pairs spanning 11 databases. We use execution accuracy as the primary metric. At test time, each method generates 8 SQL candidates with a temperature of 0.8 and self-consistency voting is used to select the final SQL that is evaluated.

Training Data. We construct several training datasets based on the same set of 4,000 "complex" samples from the SynSQL dataset used in our off-policy data generation experiments. In the on-policy setting, we use the original question–SQL pairs directly for training. The SynSQL dataset also provides chain-of-thought (CoT) reasoning traces (**SynSQL CoT**). To study the impact of different off-policy generation strategies, we additionally construct three variants of this dataset in which the original SynSQL CoT traces are replaced with the off-policy reasoning traces produced by each of the methods evaluated in our off-policy data generation experiments. These variants are denoted as **SynSQL CoT + Direct**, **SynSQL CoT + Hint Aug.**, and **SynSQL CoT + IEC**. This substitution procedure ensures that each dataset contains exactly 4,000 samples, controlling for the effect of dataset size and ensuring differences arise from trace content rather than dataset size. In all datasets, the target questions and SQL are the same, but only the off-policy reasoning target is changed.

Hyperparameters and Training Environment. During training, we set the learning rate as 1e-6, training batch size to 128, max response length as 4096, and the number of rollouts of the actor model to 8. For mixed-policy settings, one of the on-policy rollouts is replaced with the off-policy rollout. We use 8 A100 80GB GPUs for training.

6 Results

6.1 Off-Policy Data Generation

Generation Method	% Accuracy
Direct Sampling	59.4
Hint Augmented Sampling	76.5
Iterative Error Correction	**91.7**

Table 1: Accuracy of each data generation method in producing high scoring off-policy rollouts.

Accuracy for each generation method and model is reported in Table 1. We find that IEC produces a substantially higher number of correct off-policy reasoning traces than the other methods. IEC achieves over 91% accuracy in generating high-quality reasoning traces, representing a 32% improvement over Direct Sampling and a 15% improvement over Hint Augmented Sampling. IEC's iterative process of identifying errors and explicitly addressing them yields a markedly higher proportion of valid reasoning traces than approaches that rely on a static set of hints or none at all.

These results also demonstrate that hints provide a clear benefit for the baseline approaches. Generating hints prior to reasoning trace generation improves accuracy by 17% relative to the Direct Sampling method, indicating that such scaffolding can help guide the model toward valid reasoning trajectories. However, this improvement remains insufficient to match the performance of IEC, suggesting that iterative feedback and correction play a critical role in producing high-quality off-policy reasoning data.

6.2 Mixed-Policy Reinforcement Learning

Our primary results are shown in Table 2. Among models trained with the simplified mixed-policy GRPO method, the highest execution accuracy is achieved when using SynSQL CoT with IEC substitutions. In this setting, 91% of off-policy reasoning traces are generated via IEC, with the remaining 9% drawn from the original SynSQL chain-of-thought (CoT) traces. Compared to the other simplified mixed-policy variants, this configuration yields modest gains on Spider Dev (+1.25%), comparable performance on Spider Test, and substantial improvements on BIRD (+2.8%).

We observe similar trends when comparing against both the base model and the on-policy GRPO model. Relative to the base model, this approach achieves an average improvement of +2.87% across benchmarks, including a notable +6.32% gain on BIRD. Compared to on-policy GRPO, it yields an average improvement of +2.25%, with a +3.53% gain on BIRD. These results indicate that mixed-policy GRPO can significantly enhance text-to-SQL performance beyond standard on-policy training, particularly on more challenging benchmarks.

Further gains are observed when using LUFFY, a mixed-policy GRPO variant that incorporates policy shaping via regularized importance sampling to encourage exploration beyond the provided off-policy traces. In this setting, training with either the original SynSQL CoT or SynSQL CoT augmented with Direct Sampling traces achieves the highest average performance across benchmarks (73.31% and 73.36%, respectively). These correspond to improvements of approximately +4.7% over the base model and +4.06% over on-policy GRPO. As before, the largest gains occur on BIRD,

Training Method	Training Data	Spider Dev	Spider Test	BIRD Dev	Average
None	None	77.95	79.23	48.70	68.63
On-Policy	SynSQL	76.98	79.27	51.49	69.25
Mixed-Policy	SynSQL CoT	78.92	79.23	52.22	70.12
	SynSQL CoT + Direct	78.63	78.20	52.74	69.86
	SynSQL CoT + Hint Aug.	77.95	77.13	52.09	69.06
	SynSQL CoT + IEC	**80.17**	**79.32**	**55.02**	**71.50**
LUFFY	SynSQL CoT	81.91	**81.97**	**56.06**	73.31
	SynSQL CoT + Direct	**82.21**	81.88	56.00	**73.36**
	SynSQL CoT + Hint Aug.	80.85	80.30	52.54	71.23
	SynSQL CoT + IEC	79.21	80.20	53.52	70.98

Table 2: Execution accuracy comparison on the Spider and BIRD benchmarks. Models were trained using the execution-based reward function. Bold values indicate highest accuracy for each training method.

with improvements of +7.36% over the base model and +4.57% over on-policy training. Notably, all LUFFY-trained models outperform both the base and on-policy baselines across all benchmarks regardless of training data used, further underscoring the benefits of mixed-policy training.

Furthermore, our experiments highlight that the effectiveness of a given set of off-policy reasoning traces is dependent on the mixed-policy training algorithm that is used. With the simplified mixed-policy method, substituting in off-policy traces produced via IEC improves performance across Spider and BIRD when compared to using the original SynSQL chain-of-thought reasoning traces. However, these results are reversed when using LUFFY for training, where the original SynSQL chain-of-thought reasoning traces results in more capable text-to-SQL models and partial substitution with Direct Sampling traces produces comparable performance. Understanding the underlying causes of these differences is an important direction for future work and may enable the design of more effective, training-aware data generation methods.

We additionally train models using only the subset of data for which each generation method produces a valid off-policy reasoning trace. Full results are provided in Table 3 of Appendix C. Among these settings, combining Direct Sampling with the simplified mixed-policy method yields the strongest average performance across Spider and BIRD (70.68%), achieving a +1.43% improvement over the on-policy model despite using 40% fewer training samples, and a +2.05% gain over the base model. However, training with datasets that combine off-policy traces with the original SynSQL CoT remains more effective overall as shown in Table 2. This suggests that mixing reasoning traces from multiple generation methods provides complementary supervision signals, leading to stronger performance than relying on any single method alone. Exploring strategies for combining diverse off-policy generation approaches represents a promising direction for improving mixed-policy GRPO training.

7 Conclusion

In this work, we introduced mixed-policy GRPO for text-to-SQL, adapting reinforcement learning-based training to incorporate both on-policy and off-policy reasoning trajectories. We conducted a systematic study of off-policy data generation strategies, including our proposed Iterative Error Correction (IEC) method, and demonstrated that mixed-policy training improves performance over both base models and on-policy GRPO across standard text-to-SQL benchmarks.

These results point to several promising directions for future work. One avenue is the exploration of more robust reward functions, which may better capture intermediate reasoning quality and further improve training signals. It is also important to examine whether the gains observed with mixed-policy methods persist at larger model scales. Our findings additionally suggest that the effectiveness of off-policy data depends on its interaction with the specific training method employed. A deeper investigation into these interactions could inform the design of more effective data generation strategies.

References

Ahmed M Adly, Mostafa Samy, and Amr Fawzy. 2025. Gazal-r1: Achieving state-of-the-art medical reasoning with parameter-efficient two-stage training. *arXiv preprint arXiv:2506.21594*.

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, Christopher Potts, Koushik Sen, Alexandros G. Dimakis, Ion Stoica, Dan Klein, Matei Zaharia, and Omar Khattab. 2026. Gepa: Reflective prompt evolution can outperform reinforcement learning. *Preprint*, arXiv:2507.19457.

Mingzhe Du, Luu Anh Tuan, Yue Liu, Yuhao Qing, Dong Huang, Xinyi He, Qian Liu, Zejun Ma, and See-kiong Ng. 2025. Afterburner: Reinforcement learning facilitates self-improving code efficiency optimization. *arXiv preprint arXiv:2505.23387*.

Shuzheng Gao, Chaozheng Wang, Cuiyun Gao, Xiaoqian Jiao, Chun Yong Chong, Shan Gao, and Michael Lyu. 2025. The prompt alchemist: Automated llm-tailored prompt optimization for test case generation. *arXiv preprint arXiv:2501.01329*.

Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from SQL queries improves neural semantic parsing. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 1597–1607, Brussels, Belgium. Association for Computational Linguistics.

Yu Guo, Dong Jin, Shenghao Ye, Shuangwu Chen, Jian Yang, and Xiaobin Tan. 2025. SQLForge: Synthesizing reliable and diverse data to enhance text-to-SQL reasoning in LLMs. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 8441–8452, Vienna, Austria. Association for Computational Linguistics.

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Star-sql: Self-taught reasoner for text-to-sql. *arXiv preprint arXiv:2502.13550*.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems*, volume 36, pages 42330–42357. Curran Associates, Inc.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. 2025. Understanding r1-zero-like training: A critical perspective. *arXiv preprint arXiv:2503.20783*.

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*.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: iterative refinement with self-feedback. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NeurIPS '23, Red Hook, NY, USA. Curran Associates Inc.

Vaskar Nath, Elaine Lau, Anisha Gunjal, Manasi Sharma, Nikhil Baharte, and Sean Hendryx. 2025. Adaptive guidance accelerates reinforcement learning of reasoning models. *arXiv preprint arXiv:2506.13923*.

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

Reid Pryzant, Dan Iter, Jerry Li, Yin Lee, Chenguang Zhu, and Michael Zeng. 2023. Automatic prompt optimization with "gradient descent" and beam search. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7957–7968.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, and 1 others. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pages 8634–8652. Curran Associates, Inc.

Marko Sterbentz, Kevin Cushing, Cameron Barrie, and Kristian J Hammond. 2026. Ringsql: Generating synthetic data with schema-independent templates for text-to-sql reasoning models. *arXiv preprint arXiv:2601.05451*.

Bailin Wang, Wenpeng Yin, Xi Victoria Lin, and Caiming Xiong. 2021. Learning to synthesize data for

semantic parsing. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 2760–2766, Online. Association for Computational Linguistics.

Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, Ugur Cetintemel, and Carsten Binnig. 2020. Dbpal: A fully pluggable nl2sql training pipeline. In *Proceedings of the 2020 ACM SIGMOD International Conference on Management of Data*, SIGMOD '20, page 2347–2361, New York, NY, USA. Association for Computing Machinery.

Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical SQL-to-question generation for cross-domain text-to-SQL parsing. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8974–8983, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Hao Yan, Saurabh Srivastava, Yintao Tai, Sida I Wang, Wen-tau Yih, and Ziyu Yao. 2023. Learning to simulate natural language feedback for interactive semantic parsing. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 3149–3170.

Jianhao Yan, Yafu Li, Zican Hu, Zhi Wang, Ganqu Cui, Xiaoye Qu, Yu Cheng, and Yue Zhang. 2025. Learning to reason under off-policy guidance. *arXiv preprint arXiv:2504.14945*.

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.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Gaurav Nuti, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2026. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. *Preprint*, arXiv:2505.20315.

Tao Yu, Chien-Sheng Wu, Xi Victoria Lin, Bailin Wang, Yi Chern Tan, Xinyi Yang, Dragomir Radev, Richard Socher, and Caiming Xiong. 2020. Grappa: Grammar-augmented pre-training for table semantic parsing. *arXiv preprint arXiv:2009.13845*.

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.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Shiji Song, and Gao Huang. 2025. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? *arXiv preprint arXiv:2504.13837*.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D Goodman. 2024. Star: Self-taught reasoner bootstrapping reasoning with reasoning. In *Proc. the 36th International Conference on Neural Information Processing Systems*, volume 1126.

John M Zelle and Raymond J Mooney. 1996. Learning to parse database queries using inductive logic programming. In *Proceedings of the thirteenth national conference on Artificial intelligence-Volume 2*, pages 1050–1055.

Sheng Zhang, Qianchu Liu, Guanghui Qin, Tristan Naumann, and Hoifung Poon. 2025a. Med-rlvr: Emerging medical reasoning from a 3b base model via reinforcement learning. *arXiv preprint arXiv:2502.19655*.

Xiaotian Zhang, Yuan Wang, Zhaopeng Feng, Ruizhe Chen, Zhijie Zhou, Yan Zhang, Hongxia Xu, Jian Wu, and Zuozhu Liu. 2025b. Med-u1: Incentivizing unified medical reasoning in llms via large-scale reinforcement learning. *arXiv preprint arXiv:2506.12307*.

Rosie Zhao, Alexandru Meterez, Sham Kakade, Cengiz Pehlevan, Samy Jelassi, and Eran Malach. 2025. Echo chamber: Rl post-training amplifies behaviors learned in pretraining. *arXiv preprint arXiv:2504.07912*.

Victor Zhong, Mike Lewis, Sida I. Wang, and Luke Zettlemoyer. 2020. Grounded adaptation for zero-shot executable semantic parsing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6869–6882, Online. Association for Computational Linguistics.

A Prompt Templates

The prompt used when generating a set of hints at the initial iteration is shown in Figure 3.

Figure 4 shows the hint generation prompt at later iterations and includes the complete set of hints produced during the course of the previous iterations, the predicted SQL output from the previous iteration, and some updated instructions to account for these when updating the set of hints.

The prompt used to generate the off-policy reasoning traces is shown in Figure 5.

B Mixed-Policy Training Formalization

A complete formalization of the mixed-policy training based on the work of (Yan et al., 2025) is provided below.

Error Hint Generation Prompt (First Iteration)

```
The task is to reason about and determine what errors might occur when trying to
convert a question to SQL in order to produce hints that will help guide the generation
of the SQL query.

Here is the database schema with additional info: {schema}

Question: {question}

Here is the ground truth SQL that will answer the question: {ground_truth_sql}

Think through possible errors that could occur when converting the question to the
ground truth SQL and think of some hints that can be used to help guide the conversion
of the question into the ground truth SQL. Produce and enclose the reasoning process
within <think> </think> tags, and then produce and enclose the final set of hints
within <hints> </hints> tags.
```

Figure 3: The prompt used when generating the hints at the first iteration.

Let $\pi_{\theta_{\text{old}}}$ and π_θ denote the on-policy model distribution where both represent probability distributions over the possible tokens, and let π_ϕ denote the off-policy distribution. The advantage A_i for the mixed-policy GRPO is computed as follows:

$$A_i = \frac{R(\tau_i) - \text{mean}(\mathcal{G}_{\text{on}} \cup \mathcal{G}_{\text{off}})}{\text{std}(\mathcal{G}_{\text{on}} \cup \mathcal{G}_{\text{off}})} \tag{6}$$

where $\mathcal{G}_{\text{on}}$ and $\mathcal{G}_{\text{off}}$ are the sets of rewards for the on- and off-policy rollouts, respectively. These are computed as follows:

$$\mathcal{G}_{\text{on}} = \{R(\tau_i) \mid \tau_i \sim \pi_{\theta_{\text{old}}}(\tau),\ i = 1\!:\!N_{\text{on}}\} \tag{7}$$

$$\mathcal{G}_{\text{off}} = \{R(\tau_j) \mid \tau_j \sim \pi_\phi(\tau),\ j = 1\!:\!N_{\text{off}}\} \tag{8}$$

The mixed-policy objective is then computed as follows:

$$\mathcal{J}_{\text{Mixed}}(\theta) = \frac{1}{Z}\Bigg(\underbrace{\sum_{j=1}^{N_{\text{off}}}\sum_{t=1}^{|\tau_j|}\text{CLIP}(\hat{r}_{j,t}(\theta,\phi), A_j, \epsilon)}_{\text{off-policy objective}} + \underbrace{\sum_{i=1}^{N_{\text{on}}}\sum_{t=1}^{|\tau_i|}\text{CLIP}(r_{i,t}(\theta), A_i, \epsilon)}_{\text{on-policy objective}}\Bigg) \tag{9}$$

where Z is the normalization factor computed as:

$$Z = \sum_{j=1}^{N_{\text{off}}}|\tau_j| + \sum_{i=1}^{N_{\text{on}}}|\tau_i| \tag{10}$$

and $\text{CLIP}(r, A, \epsilon)$ is a surrogate objective function used to ensure updates are still within the trust region of the old policy $\pi_{\theta_{\text{old}}}$, computed as

$$\text{CLIP}(r, A, \epsilon) = \min\big[r \cdot A, \text{clip}(r; 1-\epsilon, 1+\epsilon) \cdot A\big] \tag{11}$$

and $\hat{r}_{j,t}(\theta, \phi)$ and $r_{i,t}(\theta)$ are importance sampling terms used to calibrate the gradient and computed as follows:

$$\hat{r}_{j,t}(\theta,\phi) = \frac{\pi_\theta(\tau_{j,t} \mid \tau_{j,<t})}{\pi_\phi(\tau_{j,t} \mid \tau_{j,<t})}, \tag{12}$$

$$r_{i,t}(\theta) = \frac{\pi_\theta(\tau_{i,t} \mid \tau_{i,<t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t} \mid \tau_{i,<t})} \tag{13}$$

C Training with Valid Off-Policy Traces

A full listing of results for models trained on data with valid off-policy reasoning traces is provided in Table 3.

Error Hint Generation Prompt (Successive Iterations)

```
The task is to reason about and determine what errors might occur when trying to
convert a question to SQL in order to produce hints that will help guide the generation
of the SQL query.

Here is the database schema with additional info: {schema}

Question: {question}

Here is the ground truth SQL that will answer the question: {ground_truth_sql}

Here is the set of previous hints that were identified:
{hints}

Here is the predicted SQL output that was generated for the question: {predicted_SQL}

Think through possible errors that could occur when converting the question to the
ground truth SQL and think of some hints that can be used to help guide the conversion
of the question into the ground truth SQL. Compare the incorrect predicted SQL with the
ground truth SQL to identify new errors and hints. Use the new hints and the previous
set of hints to write the final set of all hints. Produce and enclose the reasoning
process within <think> </think> tags, and then produce and enclose the final set of
hints within <hints> </hints> tags.
```

Figure 4: The prompt used when generating the hints at iterations after the first round.

Training Data Source	Training Method	Spider Dev	Spider Test	BIRD Dev	Average
Direct Sampling (2373 examples)	On-Policy	77.37	77.18	**53.06**	69.20
	Mixed-Policy	**79.50**	**79.60**	52.93	**70.68**
	LUFFY	78.05	77.97	48.83	68.28
Hint Augmented Sampling (3060 examples)	On-Policy	77.37	77.18	**53.39**	69.31
	Mixed-Policy	77.08	**79.93**	48.31	68.44
	LUFFY	**79.05**	77.97	51.37	**69.46**
IEC (3647 examples)	On-Policy	75.92	75.55	**50.78**	67.42
	Mixed-Policy	72.73	74.24	50.59	65.85
	LUFFY	**78.53**	**79.97**	48.50	**69.00**

Table 3: Execution accuracy comparison on the Spider and BIRD benchmarks for models trained on samples where each of the three data generation methods produced an off-policy reasoning trace.

SQL Generation Prompt with Error Hints

```
You are a helpful SQL expert assistant.
The assistant first thinks about how to write the SQL query by analyzing the question,
database schema, external knowledge and hints, then provides the final SQL query. The
reasoning process and SQL query are enclosed within <think> </think> and <answer>
</answer> tags respectively. The answer must contain the SQL query within ```sql ```
tags.

Database Schema: {schema}

External Knowledge: {external_knowledge}

Hints: {hints}

For example:
<think>
To translate the given natural language question into an executable SQLite query, we
need to follow these detailed steps:
1. **Identify Key Elements**: The question queries for code snippets that are both
complicated (complexity score > 5) and public (`is_public` = 1). We need to retrieve
their descriptions and complexity scores.
2. **Focus on Relevant Tables**: The `code_snippets` table contains the necessary
fields (`description`, `complexity`, `is_public`).
3. **Construct the Query**: We should select the required fields (`description` and
`complexity`) from the `code_snippets` table. We also apply the conditions specified in
the question to filter the results.
4. **Ordering**: The reference solution includes an `ORDER BY` clause to sort results
by complexity in descending order, which is a reasonable way to present the data to
highlight the most complex snippets first.
5. **Final Query Construction**: Putting all this together into a SQL query.
</think>
<answer>
Here's how the query can be written:
```sql
SELECT description, complexity FROM code_snippets WHERE complexity > 5 AND is_public =
1 ORDER BY complexity DESC;
```
This query retrieves the descriptions and complexity scores of code snippets that are
both complicated (complexity > 5) and publicly available (`is_public` = 1), sorted by
complexity in descending order.

This solution is straightforward and precisely matches the requirements of the
question. It avoids unnecessary complexities, such as joining or selecting columns not
relevant to the query itself.
</answer>

Question: {question}
```

Figure 5: The prompt used when generating the off-policy reasoning traces using hints.