Uncovering the Impact of Chain-of-Thought Reasoning for Direct Preference Optimization: Lessons from Text-to-SQL

Hanbing Liu[1]*, Haoyang Li[2,3]*, Xiaokang Zhang[2,3], Ruotong Chen[2], Haiyong Xu[5], Tian Tian[5], Qi Qi[1], Jing Zhang[2,4]†

[1]Gaoling School of Artificial Intelligence, Renmin University of China, Beijing, China,
[2]School of Information, Renmin University of China, Beijing, China,
[3]Key Laboratory of Data Engineering and Knowledge Engineering, Beijing, China,
[4]Engineering Research Center of Database and Business Intelligence, Beijing, China,
[5]China Mobile Information Technology Center
{liuhanbing, lihaoyang.cs, zhang-jing}@ruc.edu.cn

Abstract

Direct Preference Optimization (DPO) has proven effective in complex reasoning tasks like math word problems and code generation. However, when applied to Text-to-SQL datasets, it often fails to improve performance and can even degrade it. Our investigation reveals the root cause: unlike math and code tasks, which naturally integrate Chain-of-Thought (CoT) reasoning with DPO, Text-to-SQL datasets typically include only final answers (gold SQL queries) without detailed CoT solutions. By augmenting Text-to-SQL datasets with synthetic CoT solutions, we achieve, for the first time, consistent and significant performance improvements using DPO.

Our analysis shows that CoT reasoning is crucial for unlocking DPO's potential, as it mitigates reward hacking, strengthens discriminative capabilities, and improves scalability. These findings offer valuable insights for building more robust Text-to-SQL models. To support further research, we publicly release the code and CoT-enhanced datasets [1].

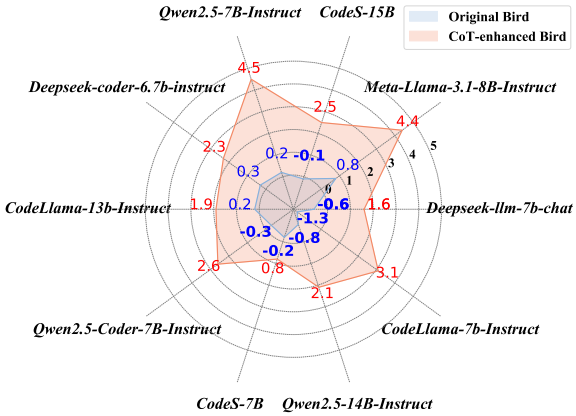


Figure 1: Performance gains achieved by DPO over SFT on 10 base LLMs (Improved Execution Accuracy under greedy decoding, %). Models SFT with the original non-CoT dataset (Blue) hardly improve in the DPO stage, while DPO's improvement is significant when SFT is done on a CoT-enhanced dataset (Orange). This shows that CoT reasoning is crucial for unlocking DPO's potential, ensuring its effectiveness and stability.

1 Introduction

Text-to-SQL has recently gained significant attention in natural language processing and database research (Li et al., 2024a; Wang et al., 2020; Fu et al., 2023; Pourreza and Rafiei, 2024a). It translates natural language questions into SQL queries, allowing non-experts to easily access data, making it a valuable tool for business intelligence, data exploration, and other data-centric applications.

With large language models (LLMs), two main approaches have emerged for solving Text-to-SQL: prompting-based methods (Talaei et al., 2024; Pourreza and Rafiei, 2024a; Pourreza et al., 2024) and supervised fine-tuning (SFT) methods (Pourreza and Rafiei, 2024b; Li et al., 2023, 2024a). Prompting-based methods often rely on powerful closed-source LLMs, making them costly and slow, and raising data privacy concerns. In contrast, SFT trains open-source, deployable LLMs using benchmark datasets like Spider and Bird. However, SFT performance is often limited by the scarcity of high-quality training data, which is expensive and time-consuming to create.

Recent studies in complex reasoning tasks, such as math word problems (Xu et al., 2024) and code generation (Zhang et al., 2024), have demonstrated that preference optimization algorithms (*e.g.*, DPO (Rafailov et al., 2024b), KTO (Ethayarajh et al., 2024), SimPO (Meng et al., 2024)) can significantly enhance SFT models. These algorithms leverage preference data pairs to enable models to distinguish between correct and incorrect

*Equal contribution.
†Corresponding author.
[1]https://github.com/RUCKBReasoning/DPO_Text2SQL

Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), pages 21223–21261
July 27 - August 1, 2025

responses, addressing the limitations of simple SFT. Despite the proven success of preference optimization techniques, recent works in Text-to-SQL have rarely adopted these methods to improve the Text-to-SQL capabilities of LLMs. This raises a critical question: *How much improvement can preference optimization bring to the Text-to-SQL task?*

Preliminary Experiments. To answer this question, we conduct initial experiments on Bird (Li et al., 2024b), a challenging cross-domain Text-to-SQL benchmark. Each data sample consists of a <question, database, SQL query> triplet. Text-to-SQL models receive the question and database information (*e.g.*, table names, column names, data types, etc.) and generate the target SQL query. To ensure the universality of our findings, we evaluate 10 open-source LLMs, ranging from 6.7B to 15B parameters. For preference optimization, we employ DPO, a widely adopted technique used in cutting-edge LLMs like LLaMA3 (Dubey et al., 2024), Qwen2.5 (Yang et al., 2024b), and Mixtral (Jiang et al., 2024).

Specifically, we follow the standard DPO training pipeline, which consists of three key steps: (1) **SFT**: The base LLM is first fine-tuned on Bird's training set. (2) **Preference Pair Construction**: Using the SFT model, multiple SQL queries are sampled for each training sample. Correct and incorrect queries are identified through database execution to create preference pairs. (3) **DPO Training**: Finally, the SFT model is further trained on these preference pairs using the DPO loss, resulting in the final DPO model.

Observations. The "Original Bird" area in Figure 1 illustrates the performance gains introduced by DPO, measured as the improvement in execution accuracy between the DPO model and the SFT model with greedy search inference. Surprisingly, the results reveal that DPO does not consistently improve performance; in fact, it leads to performance degradation for 6 out of the 10 evaluated LLMs. To make preference optimization effective for Text-to-SQL, we additionally explore several strategies, including hyperparameter tuning (Rafailov et al., 2024b), integrating SFT loss (Ouyang et al., 2022), replacing DPO with KTO (Ethayarajh et al., 2024), and using a small model to construct preference data (Yang et al., 2024c). However, as shown in Appendix A, these attempts still result in limited performance improvements ($<$1.5%).

Hypothesis. After extensive but unsuccessful algorithmic exploration, we hypothesize that the suboptimal performance of DPO in the Text-to-SQL task is primarily due to a critical yet often-overlooked factor: the quality of the data. By analyzing datasets for complex reasoning tasks, such as MATH (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), CodeUltraFeedback (Weyssow et al., 2024), Orca-Math (Mitra et al., 2024), and DART-Math (Tong et al., 2024), we observe that these datasets provide not only final answers but also chain-of-thought (CoT)-styled solutions with detailed reasoning steps. These CoT solutions bridge the gap between input questions and final answers, enabling LLMs to achieve better generalization and interpretability during SFT and DPO training. In contrast, Text-to-SQL datasets like Bird (Li et al., 2024b), Spider (Yu et al., 2018), WikiSQL (Zhong et al., 2017), and ScienceBenchmark (Zhang et al., 2023) only provide final answers (*i.e.*, gold SQL queries), forcing SFT and DPO to rely solely on SQL queries as training labels. This discrepancy leads us to propose a hypothesis: *The effectiveness of DPO is likely attributed to the use of CoT, a crucial factor that is often overlooked.*

Verification. To test this hypothesis, we introduce a pipeline to study how CoT affects DPO's performance in the Text-to-SQL task. We use an LLM-based CoT synthesizer to efficiently generate step-by-step CoT solutions for Text-to-SQL datasets with minimal human effort. The synthesizer takes the database information, question, and gold SQL query as input. Then, using the same settings as earlier experiments, we apply SFT and DPO to the CoT-enhanced Bird dataset. As shown in Figure 1, adding CoT significantly improves DPO's performance across all 10 evaluated LLMs. Additionally, we extend our evaluations to other Text-to-SQL benchmarks, including Spider, Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), Spider-Realistic (Deng et al., 2021), and Dr.Spider (Chang et al., 2023). Consistent trends are observed across these benchmarks.

Analysis. To understand why CoT reasoning is essential for unlocking DPO's potential, we conduct a comprehensive analysis and make three key observations. First, introducing CoT significantly reduces reward hacking during DPO training, ensuring stable and effective performance. Second, CoT enhances DPO's effectiveness as an implicit reward model, improving its ability to discriminate between correct and incorrect responses. Finally, CoT increases DPO's scalability, both in terms of

the number of preference data and inference-time sampling budgets.

Our contributions are summarized as follows:

- We conduct extensive experiments on Text-to-SQL datasets to evaluate DPO within the standard training pipeline. Contrary to prior studies, we find that DPO does not consistently improve performance and can sometimes degrade it. However, by augmenting these datasets with synthetic CoT solutions, we achieve stable and significant performance improvements with DPO for the first time. As existing works overlook the critical data issue in Text-to-SQL, our findings provide important insights for effectively integrating DPO into Text-to-SQL pipelines.
- We also provide a comprehensive analysis to understand why CoT reasoning is essential for DPO. Our findings reveal that incorporating CoT mitigates reward hacking, strengthens discriminative ability, and enhances scalability.

2 Related Work

Text-to-SQL. The Text-to-SQL task aims to convert natural language questions into SQL queries for a given database. With the emergence of large language models (LLMs) like GPT-4 (OpenAI, 2024a) and Gemini (Anil et al., 2023), the field has rapidly shifted towards leveraging LLMs for unified processing. These methods typically involve fine-tuning open-source language models (Li et al., 2024a; Pourreza and Rafiei, 2024b; Yang et al., 2024c) or prompting closed-source LLMs through a multi-agent framework (Pourreza and Rafiei, 2024a; Gao et al., 2024; Talaei et al., 2024; Pourreza et al., 2024; Wang et al., 2024a). Our work focuses on utilizing open-source LLMs and investigates the joint effectiveness of CoT reasoning and preference learning.

Preference Optimization. Preference optimization aims to align the model with the preferences of responses, which typically requires a compare-based training set. To this end, various methods have been proposed, from DPO (Rafailov et al., 2024b) to its variants SimPO (Meng et al., 2024), KTO (Ethayarajh et al., 2024), and IPO (Azar et al., 2024). In this study, we employ DPO as our preference optimization algorithm. We note that only one study, SENSE (Yang et al., 2024c), also employs DPO for optimizing Text-to-SQL models. However, their approach differs significantly from ours. Specifically, we follow the standard DPO pipeline, collecting preference data directly from the SFT model, whereas SENSE uses a small-scale model (1B parameters) to construct preference data for larger models, which may pose challenges in terms of transferability. Additionally, SENSE continues to use SQL queries as training labels, while this paper leverages CoT-style solutions.

Learning from Execution Feedback. Prior to the advent of LLMs, learning through execution feedback had already been widely applied to code-related tasks. CodeRL (Le et al., 2022) uses unit test results as rewards to optimize of pre-trained models through reinforcement learning. Other notable works include RLTF (Liu et al., 2023), StepCoder (Dou et al., 2024), and PseudoFeedback (Jiao et al., 2024). Nowadays, similar approaches are employed in the post-processing of general-purpose models to enhance their logical and coding capabilities (Yang et al., 2024b; DeepSeek-AI et al., 2024). However, existing general code models have not shown advantages in Text-to-SQL (Li et al., 2024a), and the substantial memory and CPU consumption of SQL execution makes online methods infeasible. We are the first to achieve stable and significant gains in Text-to-SQL by leveraging execution feedback and preference optimization.

3 Pipeline

3.1 Overview

Figure 2 illustrates our pipeline, which consists of three steps: (1) CoT synthesis, (2) supervised fine-tuning (SFT), and (3) direct preference optimization (DPO). Initially, we use an LLM as the CoT synthesizer to generate CoT solutions for a given Text-to-SQL dataset. With this CoT-enhanced dataset, we perform SFT and utilize feedback from databases to construct preference data pairs, followed by DPO. Further details are provided below.

3.2 Chain-of-Thought Synthesis

To minimize human annotation costs, we utilize an LLM as the CoT synthesizer to generate CoT solutions for existing Text-to-SQL datasets. Specifically, for each data sample <question, database, SQL qeury>, we employ GPT-4o-mini (OpenAI, 2024b) to generate K diverse CoT solutions that demonstrate the step-by-step conversion of the question into the gold SQL query. Since the final answers (*i.e.*, gold SQL queries) are provided to the CoT synthesizer, the generated CoT solu-

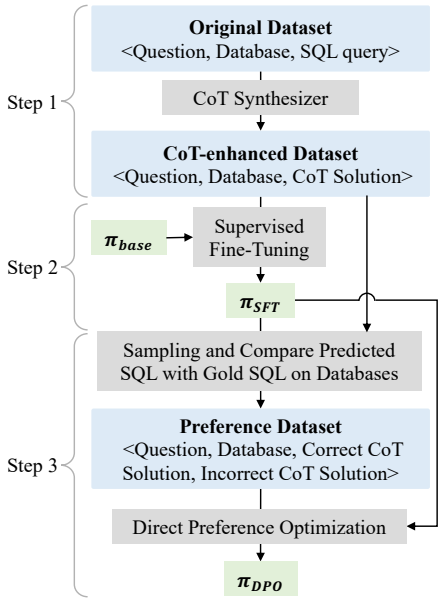


Figure 2: Overview of the proposed pipeline.

tions are both reliable and accurate, making them suitable for subsequent SFT and DPO training. The prompts used for CoT synthesis and qualitative examples are detailed in Appendix B.

3.3 Supervised Fine-tuning

Then, we perform supervised fine-tuning (SFT) using the CoT-enhanced dataset. The input prompt includes not only the question but also the database prompt, including information such as table names, column names, column data types, primary and foreign key relationships, etc. Details about the prompt construction can be found in Appendix E. The output sequences for SFT are LLM-synthesized CoT solutions. Formally, we denote the question as q, the database prompt as d, and the output CoT solution as c. The objective of SFT is guided by a conditional next-token prediction loss:

$$\mathcal{L}_{\mathcal{SFT}} = -\mathbb{E}_{(q,d,c)\sim D_S}[\log \pi_{base}(c \mid q, d)],$$

where D_S denotes the CoT-enhanced training set, and π_{base} refers the base model. After fine-tuning, we obtain model π_{SFT}, serving as the reference model for the subsequent DPO.

3.4 Direct Preference Optimization

Then, we apply DPO (Rafailov et al., 2024b) to further improve the Text-to-SQL capabilities of the SFT model. A brief overview of the DPO algorithm, including its learning objective, implicit reward mechanism, and token-level credit assignment, is provided in Appendix C.

Preference Dataset Construction. DPO requires a preference dataset to teach the model to distinguish between correct and incorrect responses. In this study, each preference data sample consists of a quadruple: <question, database, correct CoT solution, incorrect CoT solution>. To construct this dataset, we sample CoT solutions from the supervised fine-tuned (SFT) model and use database feedback to determine the correctness of each sampled CoT solution. Specifically, for each data sample in the training set, we generate N distinct CoT solutions from the reference model π_{SFT}. We then extract predicted SQL queries from these solutions using regular expressions and execute both the predicted and gold SQL queries on the corresponding databases. A sampled CoT solution is labeled as correct only if the execution results (*e.g.*, records that satisfy certain filter conditions) of the predicted SQL query completely match those of the gold SQL query; otherwise, it is labeled as incorrect. For data samples with multiple correct and incorrect CoT solutions, we randomly select one correct and one incorrect solution to form a preference pair for DPO training.

DPO Learning Objective. The goal of DPO is to maximize the margin between the log-likelihood of the correct and incorrect responses while ensuring the model remains close with the reference policy. Formally, for a question q and its corresponding database prompt d, let c^+ and c^- represent the correct and incorrect CoT solutions, respectively. The DPO loss is defined as:

$$\mathcal{L}_{\mathcal{DPO}} = -\mathbb{E}_{(q,d,c^+,c^-)\sim D_P} \log \sigma \left(\beta \log \frac{\pi_{DPO}(c^+ \mid q, d)}{\pi_{SFT}(c^+ \mid q, d)} - \beta \log \frac{\pi_{DPO}(c^- \mid q, d)}{\pi_{SFT}(c^- \mid q, d)} \right),$$

where $\mathcal{D}_P$ is the preference dataset, $\sigma(\cdot)$ is the sigmoid function, β is a hyperparameter that controls the penalty strength imposed by the KL divergence, π_{DPO} represents the DPO model, which is initialized from the SFT model at the start of training.

4 Experiment Setup

4.1 Datasets

Common Benchmark: Spider (Yu et al., 2018) is a widely used Text-to-SQL dataset comprising a training set of 7,000 samples and a development set of 1,034 samples. This dataset covers 200 databases across 138 diverse domains.

Challenging Benchmark: Bird (Li et al., 2024b) presents a more challenging benchmark,

featuring a training set of 9,428 samples and a development set of 1,534 samples. It includes 95 large databases across 37 professional domains. In contrast to Spider, Bird offers a more realistic scenario that aligns with real-world applications.

Robustness Benchmarks: Spider-DK (Gan et al., 2021b), Spider-Syn (Gan et al., 2021a), and Spider-Realistic (Deng et al., 2021) are three widely adopted robustness evaluation sets that modify the development set of Spider to simulate real-world scenarios. Another significant derivative, Dr.Spider (Chang et al., 2023), creates 17 distinct robustness evaluation sets by comprehensively perturbing the Spider development set across 3 aspects: questions, databases, and SQL queries.

4.2 Evaluation Metrics

For all benchmarks, we use the execution accuracy (EX) metric (Yu et al., 2018) to evaluate the accuracy of the model's predictions. EX measures whether the predicted and gold SQL queries produce identical execution results on the given database. For Spider's development set, we additionally employ a more robust metric, test-suite accuracy (TS) (Zhong et al., 2020), which extends EX by evaluating whether the predicted SQL query consistently passes the EX evaluation across multiple test-suite database instances.

4.3 Inference Strategy

Given a Text-to-SQL model, we explore three inference strategies: **(a) Greedy**: Use greedy decoding with a temperature of 0 to generate a response. **(b) Pass@1**: Sample a response with a temperature of 1.0. To ensure stability, we repeat this process 16 times and report the average scores. **(c) Maj@K**: Sample K responses with a temperature of 1.0 and conduct majority voting based on the execution results of the predicted SQL queries. The final prediction is selected from the most-voted group.

4.4 Implementation Details

We select 10 base models from various model families, including Deepseek (Bi et al., 2024; Guo et al., 2024), Qwen (Yang et al., 2024a; Hui et al., 2024), Llama (Dubey et al., 2024; Roziere et al., 2023), and CodeS (Li et al., 2024a). These models cover different specialties (general-purpose, code- or SQL-specific) and range from 6.7B to 15B parameters. For each LLM, we conduct SFT and DPO using either the original training dataset (Vanilla) or the CoT-enhanced dataset (Syn CoT). More implementation details are listed in Appendix D. Details about training data can be found in Appendix E.

5 Experimental Results

5.1 Main Results

The results on the Bird benchmark are presented in Table 1. Results on the Spider benchmark and its robustness variants are deferred to Appendix G.

Vanilla Models Struggle to Achieve Performance Gains in the DPO Stage. For Greedy and Maj@16, DPO gains are minimal or even negative, and improvements for the Pass@1 strategy are marginal as well. Models showing performance degradation after DPO tend to worsen as training progresses, indicating that directly applying DPO in the vanilla setting may impair performance.

Models with Synthetic CoT Achieve Stable and Significant Gains in the DPO Stage. These gains are evident across all base models and inference strategies. Even when CoT models outperform vanilla models in the SFT stage, the performance gains from CoT remain consistently significant during the subsequent DPO stage. Moreover, this phenomenon persists even when replacing GPT-4o-mini with much weaker LLMs (e.g., Qwen2.5-1.5B-Instruct) to synthesize CoT solutions, confirming the benefit of the CoT solution style per se for DPO, as discussed in Appendix F.

Synthesized CoT plus DPO Exhibit Higher Performance Ceilings. As shown in the ΔEX column in Table 1, all base models trained with CoT-enhanced data through the SFT and DPO pipeline achieved higher performance ceilings. This indicates that integrating CoT synthesis with DPO is highly effective for the Text-to-SQL task, offering a promising new approach to developing improved Text-to-SQL models. With our straightforward pipeline, Qwen2.5-14B-Instruct achieves the second-best performance on the Bird development set among all open-source models, despite having significantly fewer parameters, as shown in Table 2.

5.2 Error Analysis

To understand how DPO affects model generation and identify areas where CoT enhances DPO, we meticulously analyze errors made by Qwen2.5-7B-Instruct using the greedy decoding strategy. As shown in Table 3, errors are classified into our predefined categories, with correction rates before and after DPO presented for each. Full explanations

	Model	Bird Dev						
		Greedy		Pass@1		Maj@16		ΔEX
		SFT	DPO	SFT	DPO	SFT	DPO	
General Models								
Vanilla	Deepseek-llm-7b-chat	51.8	51.2 (-0.6)	47.9	49.1 (+1.3)	54.5	54.3 (-0.3)	-
	Meta-Llama-3.1-8B-Instruct	59.0	59.8 (+0.8)	56.1	57.2 (+1.1)	61.4	60.8 (-0.6)	-
	Qwen2.5-7B-Instruct	58.8	59.0 (+0.2)	55.1	55.7 (+0.6)	61.4	60.6 (-0.8)	-
	Qwen2.5-14B-Instruct	64.3	63.5 (-0.8)	62.3	62.6 (+0.3)	64.6	65.1 (+0.5)	-
Syn CoT	Deepseek-llm-7b-chat	54.3	55.9 (+1.6)	51.9	54.8 (+2.9)	59.1	61.0 (+1.9)	54.5 → 61.0 (**+6.5**)
	Meta-Llama-3.1-8B-Instruct	56.8	61.2 (+4.4)	57.5	59.0 (+1.5)	60.2	61.9 (+1.7)	61.4 → 61.9 (**+0.5**)
	Qwen2.5-7B-Instruct	57.4	61.9 (+4.5)	54.8	59.2 (+4.4)	63.0	64.9 (+1.9)	61.4 → 64.9 (**+3.5**)
	Qwen2.5-14B-Instruct	63.2	65.3 (+2.1)	61.8	64.7 (+2.9)	65.4	67.1 (+1.7)	64.6 → 67.1 (**+2.5**)
Coder Models								
Vanilla	Deepseek-coder-6.7b-instruct	60.6	60.9 (+0.3)	56.9	58.8 (+1.9)	59.8	61.0 (+1.2)	-
	CodeLlama-7b-Instruct-hf	57.0	55.7 (-1.3)	54.3	55.5 (+1.2)	59.1	58.5 (-0.6)	-
	CodeLlama-13b-Instruct-hf	60.0	60.2 (+0.2)	56.7	57.9 (+1.2)	61.9	62.0 (+0.1)	-
	Qwen2.5-Coder-7B-Instruct	61.6	61.3 (-0.3)	59.4	60.6 (+1.2)	61.3	62.7 (+1.4)	-
Syn CoT	Deepseek-coder-6.7b-instruct	61.5	63.8 (+2.3)	59.9	62.3 (+4.5)	64.3	65.4 (+1.1)	59.8 → 65.4 (**+5.6**)
	CodeLlama-7b-Instruct-hf	58.2	61.3 (+3.1)	56.9	60.4 (+3.5)	60.2	61.9 (+1.7)	59.1 → 61.9 (**+2.8**)
	CodeLlama-13b-Instruct-hf	62.0	63.9 (+1.9)	59.8	62.5 (+2.7)	63.6	65.8 (+2.2)	61.9 → 65.8 (**+3.9**)
	Qwen2.5-Coder-7B-Instruct	60.8	63.4 (+2.6)	59.1	62.8 (+3.7)	62.5	64.1 (+1.6)	61.3 → 64.1 (**+2.8**)
SQL-Specialized Models								
Vanilla	CodeS-7B	56.8	56.6 (-0.2)	53.7	54.6 (+0.9)	58.1	58.0 (-0.1)	-
	CodeS-15B	58.3	58.2 (-0.1)	55.6	56.2 (+0.6)	60.2	59.1 (-1.1)	-
Syn CoT	CodeS-7B	56.7	57.5 (+0.8)	54.2	55.3 (+1.1)	60.2	61.7 (+1.5)	58.1 → 61.7 (**+2.6**)
	CodeS-15B	58.6	61.1 (+2.5)	56.6	60.5 (+3.9)	62.4	63.2 (+0.8)	60.2 → 63.2 (**+3.0**)

Table 1: Model performance on the Bird development set. **Vanilla**: SFT and DPO on the original Bird training set; **Syn CoT**: SFT and DPO on the CoT-enhanced training set; **ΔEX**: The performance difference in EX between "Syn CoT + DPO" and "Vanilla + SFT" when using the same base model.

Model	Date	Size	EX
ExSL+granite-34b-code	Oct 27, 2024	34B	72.43
Qwen2.5-14B-Instruct + Syn CoT + DPO (Ours)	**Nov 12, 2024**	**14B**	**67.10**
XiYanSQL-QwenCoder-32B	Jan 09, 2025	32B	67.01
MSL-SQL+DeepSeek-V2.5	Oct 10, 2024	236B	66.82
ExSL+granite-20b-code	May 14, 2024	20B	65.38
SFT CodeS-15B	Oct 12, 2023	15B	58.47

Table 2: Comparing with top open-source model-based Text-to-SQL methods on Bird development set.

and examples of our classification criteria, along with detailed error statistics, are available in Appendix I. We find that, except for Syntax Errors, synthesized CoT improves DPO's ability to correct errors across all other categories.

DPO Excels at Correcting Errors Caused by Ignoring Details. Error types with a correction rate exceeding 25% during DPO are highlighted in bold. These errors primarily stem from the model's failure to pay sufficient attention to details. For instance, NULL/DISTINCT (Fix 40.0% with CoT) errors arise when the model overlooks missing or duplicate values in the relevant columns, leading to incorrect query results. Similarly, Column Sequence (Fix 42.9% with CoT) errors occur when the model does not return columns in the order specified in the question.

CoT Provides Logical Guidance to Enhance DPO's Error Correction. Error types with significant improvement often benefit from CoT's reasoning nature. For example, JOIN errors (Fix 32.1%, +16.5% with CoT) frequently occur when the required information resides in two tables that need to be joined via a third intermediate table. Without CoT, the model might attempt to directly join the two tables on an incorrect key. CoT's step-by-step logic clarifies these joins, aiding DPO in error correction. Conversely, for intuitive errors like Hallucination (Fix 27.2%, +3.5% with CoT) and Date (Fix 30.4%, +7.3% with CoT), CoT's impact is smaller. These errors stem from mismatches between SQL queries and database information, which vanilla DPO already can address effectively. More analysis about DPO's effect with or without

Category	Description	Type	Vanilla DPO Fix (%)	Syn CoT DPO Fix (%)	Δ(%)
External Knowledge	Neglect of hints	**[A1] EK**	0.0 (0/3)	**37.5 (3/8)**	+37.5
Schema Linking	Fails to match the question with its concerning table and columns	[B1] Table	13.0 (12/92)	15.9 (11/69)	+2.9
		[B2] JOIN	15.6 (12/77)	**32.1 (18/56)**	+16.5
		[B3] Column	10.3 (7/68)	16.1 (10/62)	+5.8
		[B4] Hallucination	23.7 (14/59)	**27.2 (28/102)**	+3.5
		[B5] Condition	16.7 (10/60)	23.2 (16/69)	+6.5
		[B6] NULL/DISTINCT	9.7 (3/31)	**40.0 (12/30)**	+30.3
Value Retrieval	Mismatch of condition with its storage format	[C1] String/Number	4.5 (1/22)	21.1 (4/19)	+16.6
		[C2] Date	23.1 (6/26)	**30.4 (7/23)**	+7.3
Operation	Misunderstands required operation in the question.	[D1] Mathematical Formula	13.3 (6/45)	18.2 (8/44)	+4.9
		[D2] Aggregation	6.7 (5/75)	18.2 (12/66)	+11.5
		[D3] Complex Operation	5.6 (1/18)	12.5 (3/24)	+6.9
Information	Fails to organize information in the right way	[E1] Redundant/Incomplete	11.8 (4/34)	19.2 (5/26)	+7.4
		[E2] Column Sequence	0 (0/5)	**42.9 (3/7)**	+42.9
		[E3] ORDER BY/LIMIT	9.1 (1/11)	12.5 (1/8)	+3.4
		[E4] Format	**66.7 (2/3)**	**33.3 (2/6)**	-33.4
Syntax Error	Inexecutatble SQL	[F1] Syntax	14.3 (2/14)	13.3 (2/15)	-1.0

Table 3: Comparison of Vanilla and Syn CoT DPO correction capability across error types on Bird development set (greedy). The base model is Qwen2.5-7B-Instruct.

CoT reasoning can be found in Appendix I.

6 Why Does DPO Benefit From CoT?

To further investigate why CoT reasoning is essential for DPO, we conduct a series of analyses using Qwen2.5-7B-Instruct on the Bird benchmark.

6.1 Better Discriminative Ability

We design an evaluation preference dataset to compare the discriminative capabilities of Vanilla and Syn CoT models after DPO. Construction details are in Appendix J. During DPO training, we assess the model's ability to select the correct response by classification accuracy based on implicit reward. The results in Figure 3 show that **CoT reasoning enables DPO models to achieve more stable and superior discriminative ability**.

Model	SFT	$\beta = 0.05$	$\beta = 0.1$	$\beta = 0.2$
Vanilla	58.8	57.6 (**-1.2**)	59.0 (**+0.2**)	57.8 (**-1.0**)
Syn CoT	57.4	61.6 (**+4.2**)	61.9 (**+4.5**)	61.9 (**+4.5**)

Table 4: DPO performance across different β values (greedy). The base model is Qwen2.5-7B-Instruct.

6.2 Better Training Stability

From a different perspective, we examine the discrepancy between the models' self-evaluation rewards for their generated outputs and the actual execution accuracy on the development set. Specifically, at the end of each epoch during DPO training, we sample responses using the current checkpoint and calculate the average implicit reward, as defined in Appendix C.2. Figure 4 illustrates the results for both the Syn CoT and Vanilla models.

Vanilla Model is Susceptible to Reward Hacking. As shown in Figure 4 (a), during DPO training, the performance of the Vanilla model initially peaks but then drops. Despite this drop, its self-reward scores continue to rise, indicating that DPO's underlying reward model mistakenly believes its outputs are improving, thereby demonstrating the phenomenon of reward hacking.

In Contrast, The Self-reward of Syn CoT Model Reflects Its Actual Performance. As shown in Figure 4 (b), In the early stages of DPO training, as the model's capabilities improve, its self-reward scores also increase. In the later stages, when the model's performance saturates, the self-reward scores stabilize rather than exhibiting the reward-hacking observed in the Vanilla model. This suggests that the Syn CoT model can recognize when its outputs are no longer improving and avoids overestimating its performance.

Output Statistics. We compare the statistical changes in the outputs of the two models (Vanilla vs. Syn CoT) after DPO in Table 5. For the Vanilla model, the average output length increased significantly (+26.6%) after DPO, accompanied by a substantial rise in the proportion of invalid SQL queries (+17.6%). In contrast, the Syn CoT model exhibited minimal changes in SQL length (+2%), and a reduced non-executable rate (-2.5%).

A Case of Reward Hacking in Vanilla + DPO.

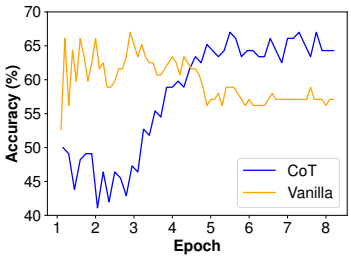
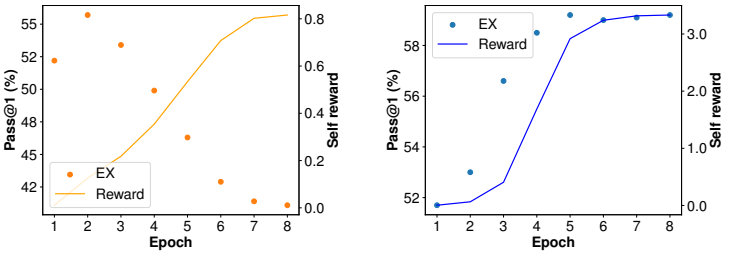
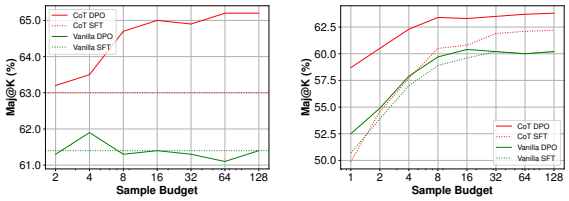


Figure 3: Comparison of model's discriminative ability during DPO (measured by classification accuracy on curated evaluation set).



(a) Vanilla (b) Syn CoT

Figure 4: Comparison of model's self-assessed performance (average implicit reward policy model given to its own roll-outs) and real performance (EX) on Bird development set (Pass@1) during DPO training.



(a) Preference Data (b) Inference-time

Figure 5: Model performance with different sample budget K in each stage (Maj@K). Qwen2.5-7B-Instruct is used as the base model.

Metric	Vanilla		Syn CoT	
	SFT	DPO	SFT	DPO
SQL Character Number	173	219	178	182
Non-executable SQL(%)	2.0	19.6	7.6	5.1

Table 5: Statistics on predicted SQL queries.

Furthermore, we show a classical non-executable output generated by the Vanilla DPO model in Figure 6. In this example, the model assigns an exceptionally high reward to an incorrect token, even though this token does not appear in the training dataset. This indicates a clear occurrence of reward hacking. Other prominent reward hacking patterns are presented in Appendix K.

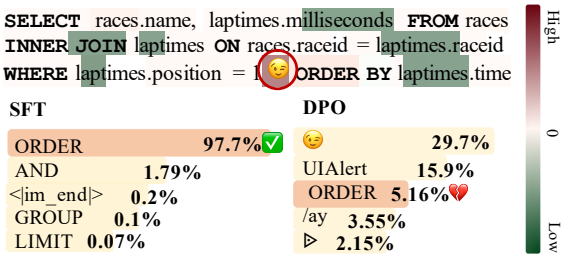


Figure 6: A case of reward hacking from Bird development set. The background color indicates token rewards.

Syn CoT Enhances DPO Robustness to Hyperparameter β. As shown in Table 4, varying the hyperparameter β reveals that the DPO training of the vanilla model is highly sensitive to changes in β. In contrast, the Syn CoT model demonstrates notable robustness, consistently delivering strong performance despite these perturbations.

6.3 Better Scalability

In this section, we discuss the scalability of our proposed pipeline to provide guidance for best practices. We report the Maj@K results in Figure 5, and results for Greedy and Pass@16 are provided in Appendix L. **Preference Data Collection (Figure 5a).** For vanilla models, collecting additional preference data does not enhance performance during the DPO stage. In contrast, increasing the sample budget for Syn CoT models consistently results in continuous performance improvements with DPO. **Inference-time Computation (Figure 5b).** By increasing the inference budget, both vanilla and Syn CoT models show improved performance, although the gains eventually stabilize. When the budget is limited, DPO offers a significant advantage for Syn CoT models over Vanilla models. For instance, "CoT + DPO" with 2 samples achieves the same performance as the "Vanilla + DPO" with 16 samples.

7 Prospects

Our trained DPO model can be integrated into existing multi-agent Text-to-SQL frameworks. As a proof of concept, we adopt Qwen2.5-7B-Instruct as the base model and integrate it into two Text-to-SQL frameworks: DTS-SQL (Pourreza and Rafiei, 2024b) and C3-SQL (Dong et al., 2023). Implementation details are provided in Appendix M. Results in Table 6 demonstrate that our model could improve performance over existing frameworks.

Bird		Spider	
Method	**EX**	**Method**	**EX**
DTS-SQL	47.7	C3-SQL	80.5
+ Syn CoT + DPO	55.9 (**+8.2**)	+ Syn CoT + DPO	81.9 (**+1.4**)

Table 6: Performance improvements of two Text-to-SQL frameworks when incorporating our trained models.

8 Conclusion

In this work, we demonstrate, for the first time, consistent and effective performance improvements using DPO on the Text-to-SQL task, enabled by synthetic CoT solutions. Through comprehensive experiments and detailed analyses, we show that CoT reasoning is essential for unlocking DPO's potential in Text-to-SQL, as it mitigates reward hacking, enhances the model's discriminative capabilities, and improves scalability during DPO training. We believe these findings will inspire researchers and practitioners to develop more robust and effective open-source Text-to-SQL models. Moreover, we also believe that our observations provide concrete evidence for the crucial impact of CoT on RL-like algorithms in complex reasoning tasks in general.

9 Limitations

The databases provided by the Bird dataset are typically large, leading to significant time consumption when executing SQL queries. This slows down the collection of preference data, as all sampled SQL queries must be executed on the databases to obtain feedback. To address this, we modify Bird's evaluation script to enable parallel execution of SQL queries using multi-processing, significantly accelerating the preference data construction process. However, resource contention among multiple processes can lead to SQL execution timeouts, causing correct predicted SQL queries to be incorrectly classified as incorrect, thereby introducing false negatives. This inaccuracy in feedback signals could potentially impact DPO training.

In contrast, the Spider dataset presents a different challenge due to the simplicity of its database values. Relying on execution results to distinguish between correct and incorrect SQL queries can lead to false positives (Zhong et al., 2020), resulting in potentially unreliable feedback signals for DPO training.

Acknowledgments

This work is supported by the National Key Research & Develop Plan (2023YFF0725100) and the National Natural Science Foundation of China (62322214, U23A20299, U24B20144, 62172424, 62276270).

References

Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M. Dai, Anja Hauth, Katie Millican, David Silver, Slav Petrov, Melvin Johnson, Ioannis Antonoglou, Julian Schrittwieser, Amelia Glaese, Jilin Chen, Emily Pitler, Timothy P. Lillicrap, Angeliki Lazaridou, Orhan Firat, James Molloy, Michael Isard, Paul Ronald Barham, Tom Hennigan, Benjamin Lee, Fabio Viola, Malcolm Reynolds, Yuanzhong Xu, Ryan Doherty, Eli Collins, Clemens Meyer, Eliza Rutherford, Erica Moreira, Kareem Ayoub, Megha Goel, George Tucker, Enrique Piqueras, Maxim Krikun, Iain Barr, Nikolay Savinov, Ivo Danihelka, Becca Roelofs, Anaïs White, Anders Andreassen, Tamara von Glehn, Lakshman Yagati, Mehran Kazemi, Lucas Gonzalez, Misha Khalman, Jakub Sygnowski, and et al. 2023. Gemini: A family of highly capable multimodal models. *CoRR*, abs/2312.11805.

Arian Askari, Christian Poelitz, and Xinye Tang. 2024. MAGIC: generating self-correction guideline for in-context text-to-sql. *CoRR*, abs/2406.12692.

Mohammad Gheshlaghi Azar, Zhaohan Daniel Guo, Bilal Piot, Rémi Munos, Mark Rowland, Michal Valko, and Daniele Calandriello. 2024. A general theoretical paradigm to understand learning from human preferences. In *International Conference on Artificial Intelligence and Statistics, 2-4 May 2024, Palau de Congressos, Valencia, Spain*, volume 238 of *Proceedings of Machine Learning Research*, pages 4447–4455. PMLR.

Xiao Bi, Deli Chen, Guanting Chen, Shanhuang Chen, Damai Dai, Chengqi Deng, Honghui Ding, Kai Dong, Qiushi Du, Zhe Fu, et al. 2024. Deepseek llm: Scaling open-source language models with longtermism. *arXiv preprint arXiv:2401.02954*.

Hasan Alp Caferoglu and Özgür Ulusoy. 2024. E-SQL: direct schema linking via question enrichment in text-to-sql. *CoRR*, abs/2409.16751.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, Steve Ash, William Yang Wang, Zhiguo Wang, Vittorio Castelli, Patrick Ng, and Bing Xiang. 2023. Dr.spider: A diagnostic evaluation benchmark towards text-to-sql robustness. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Changyu Chen, Zichen Liu, Chao Du, Tianyu Pang, Qian Liu, Arunesh Sinha, Pradeep Varakantham, and Min Lin. 2024. Bootstrapping language models with dpo implicit rewards. *Preprint*, arXiv:2406.09760.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *CoRR*, abs/2110.14168.

Tri Dao. 2023. Flashattention-2: Faster attention with better parallelism and work partitioning. *arXiv preprint arXiv:2307.08691*.

DeepSeek-AI, Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y. Wu, Yukun Li, Huazuo Gao, Shirong Ma, Wangding Zeng, Xiao Bi, Zihui Gu, Hanwei Xu, Damai Dai, Kai Dong, Liyue Zhang, Yishi Piao, Zhibin Gou, Zhenda Xie, Zhewen Hao, Bingxuan Wang, Junxiao Song, Deli Chen, Xin Xie, Kang Guan, Yuxiang You, Aixin Liu, Qiushi Du, Wenjun Gao, Xuan Lu, Qinyu Chen, Yaohui Wang, Chengqi Deng, Jiashi Li, Chenggang Zhao, Chong Ruan, Fuli Luo, and Wenfeng Liang. 2024. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *CoRR*, abs/2406.11931.

Xiang Deng, Ahmed Hassan, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350.

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*.

Shihan Dou, Yan Liu, Haoxiang Jia, Enyu Zhou, Limao Xiong, Junjie Shan, Caishuang Huang, Xiao Wang, Xiaoran Fan, Zhiheng Xi, Yuhao Zhou, Tao Ji, Rui Zheng, Qi Zhang, Tao Gui, and Xuanjing Huang. 2024. Stepcoder: Improving code generation with reinforcement learning from compiler feedback. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages 4571–4585. Association for Computational Linguistics.

Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*.

Kawin Ethayarajh, Winnie Xu, Niklas Muennighoff, Dan Jurafsky, and Douwe Kiela. 2024. KTO: model alignment as prospect theoretic optimization. *CoRR*, abs/2402.01306.

Han Fu, Chang Liu, Bin Wu, Feifei Li, Jian Tan, and Jianling Sun. 2023. Catsql: Towards real world natural language to SQL applications. *Proc. VLDB Endow.*, 16(6):1534–1547.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931.

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

Naman Goyal, Jingfei Du, Myle Ott, Giri Anantharaman, and Alexis Conneau. 2021. Larger-scale transformers for multilingual masked language modeling. In *Proceedings of the 6th Workshop on Representation Learning for NLP (RepL4NLP-2021)*, pages 29–33.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. 2024. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Proceedings of the Neural Information Processing Systems Track on Datasets and Benchmarks 1, NeurIPS Datasets and Benchmarks 2021, December 2021, virtual*.

Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron C. Courville, Alessandro Sordoni, and Rishabh Agarwal. 2024. V-star: Training verifiers for self-taught reasoners. *CoRR*, abs/2402.06457.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Albert Q. Jiang, Alexandre Sablayrolles, Antoine Roux, Arthur Mensch, Blanche Savary, Chris Bamford, Devendra Singh Chaplot, Diego de Las Casas, Emma Bou Hanna, Florian Bressand, Gianna Lengyel, Guillaume Bour, Guillaume Lample, Lélio Renard Lavaud, Lucile Saulnier, Marie-Anne Lachaux, Pierre Stock, Sandeep Subramanian, Sophia Yang, Szymon Antoniak, Teven Le Scao,

Théophile Gervet, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2024. Mixtral of experts. *CoRR*, abs/2401.04088.

Fangkai Jiao, Geyang Guo, Xingxing Zhang, Nancy F. Chen, Shafiq Joty, and Furu Wei. 2024. Preference optimization for reasoning with pseudo feedback. *CoRR*, abs/2411.16345.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles*, pages 611–626.

Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Rewardbench: Evaluating reward models for language modeling. *Preprint*, arXiv:2403.13787.

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. In *Advances in Neural Information Processing Systems 35: Annual Conference on Neural Information Processing Systems 2022, NeurIPS 2022, New Orleans, LA, USA, November 28 - December 9, 2022*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.

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.

Jiate Liu, Yiqin Zhu, Kaiwen Xiao, Qiang Fu, Xiao Han, Wei Yang, and Deheng Ye. 2023. RLTF: reinforcement learning from unit test feedback. *Trans. Mach. Learn. Res.*, 2023.

Yinhan Liu. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*, 364.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *CoRR*, abs/2405.14734.

Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. 2018. Mixed precision training. *Preprint*, arXiv:1710.03740.

Arindam Mitra, Hamed Khanpour, Corby Rosset, and Ahmed Awadallah. 2024. Orca-math: Unlocking the potential of slms in grade school math. *CoRR*, abs/2402.14830.

OpenAI. 2024a. Gpt-4 turbo and gpt-4. Https://platform.openai.com/docs/models/gpt-4-turbo-and-gpt-4.

OpenAI. 2024b. Gpt-4o mini: advancing cost-efficient intelligence. Https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv preprint arXiv:2410.01943*.

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

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

Rafael Rafailov, Joey Hejna, Ryan Park, and Chelsea Finn. 2024a. From r to q^*: Your language model is secretly a q-function. *arXiv preprint arXiv:2404.12358*.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2024b. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36.

Jeff Rasley, Samyam Rajbhandari, Olatunji Ruwase, and Yuxiong He. 2020. Deepspeed: System optimizations enable training deep learning models with over 100 billion parameters. In *Proceedings of the 26th ACM SIGKDD International Conference on Knowledge Discovery & Data Mining*, pages 3505–3506.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Zayne Sprague, Fangcong Yin, Juan Diego Rodriguez, Dongwei Jiang, Manya Wadhwa, Prasann Singhal, Xinyu Zhao, Xi Ye, Kyle Mahowald, and Greg Durrett. 2024. To cot or not to cot? chain-of-thought helps mainly on math and symbolic reasoning. *Preprint*, arXiv:2409.12183.

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

Yuxuan Tong, Xiwen Zhang, Rui Wang, Ruidong Wu, and Junxian He. 2024. Dart-math: Difficulty-aware rejection tuning for mathematical problem-solving. In *Advances in Neural Information Processing Systems 38: Annual Conference on Neural Information Processing Systems 2024, NeurIPS 2024, Vancouver, BC, Canada, December 10 - 15, 2024*.

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.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. 2024a. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Dingzirui Wang, Longxu Dou, Xuanliang Zhang, Qingfu Zhu, and Wanxiang Che. 2024b. DAC: decomposed automation correction for text-to-sql. *CoRR*, abs/2408.08779.

Martin Weyssow, Aton Kamanda, and Houari A. Sahraoui. 2024. Codeultrafeedback: An llm-as-a-judge dataset for aligning large language models to coding preferences. *CoRR*, abs/2403.09032.

Yifan Xu, Xiao Liu, Xinghan Liu, Zhenyu Hou, Yueyan Li, Xiaohan Zhang, Zihan Wang, Aohan Zeng, Zhengxiao Du, Wenyi Zhao, Jie Tang, and Yuxiao Dong. 2024. Chatglm-math: Improving math problem-solving in large language models with a self-critique pipeline. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, pages 9733–9760. Association for Computational Linguistics.

An Yang, Baosong Yang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Zhou, Chengpeng Li, Chengyuan Li, Dayiheng Liu, Fei Huang, et al. 2024a. Qwen2 technical report. *arXiv preprint arXiv:2407.10671*.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, Huan Lin, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jingren Zhou, Junyang Lin, Kai Dang, Keming Lu, Keqin Bao, Kexin Yang, Le Yu, Mei Li, Mingfeng Xue, Pei Zhang, Qin Zhu, Rui Men, Runji Lin, Tianhao Li, Tingyu Xia, Xingzhang Ren, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yu Wan, Yuqiong Liu, Zeyu Cui, Zhenru Zhang, and Zihan Qiu. 2024b. Qwen2.5 technical report. *CoRR*, abs/2412.15115.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024c. 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.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 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.

Dylan Zhang, Shizhe Diao, Xueyan Zou, and Hao Peng. 2024. PLUM: preference learning plus test cases yields better code language models. *CoRR*, abs/2406.06887.

Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to SQL systems. *Proc. VLDB Endow.*, 17(4):685–698.

Yaowei Zheng, Richong Zhang, Junhao Zhang, Yanhan Ye, Zheyan Luo, Zhangchi Feng, and Yongqiang Ma. 2024. Llamafactory: Unified efficient fine-tuning of 100+ language models. *arXiv preprint arXiv:2403.13372*.

Han Zhong, Guhao Feng, Wei Xiong, Xinle Cheng, Li Zhao, Di He, Jiang Bian, and Liwei Wang. 2024. Dpo meets ppo: Reinforced token optimization for rlhf. *arXiv preprint arXiv:2404.18922*.

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 2020, Online, November 16-20, 2020*, pages 396–411. Association for Computational Linguistics.

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

A Preliminary Experiments on DPO Tricks

Before exploring chain-of-thought reasoning, we extensively experiment with applying various DPO tricks to the original Bird dataset. This included hyperparameter tuning, using different loss variants, and exploring alternative preference data construction strategies. Specifically, for hyperparameter tuning, we evaluate different values of the β parameter in DPO, testing 0.05 and 0.2 (the default value is 0.1). For loss variants, we experiment with augmenting the DPO loss by incorporating the SFT loss for correct responses (i.e., the correct sampled SQL queries). Additionally, we explore replacing DPO with its variant, KTO (Ethayarajh et al., 2024). For preference data construction strategies, we follow SENSE (Yang et al., 2024c) to fine-tune a small-scale language model, Deepseek-coder-1.3b-instruct, on Bird's training set to collect preference data. The results, summarized in Table 7, show that none of these approaches yield significant performance improvements for DPO.

DPO Tricks (SFT= 58.8%)	
`Hyper-parameter Tuning`	
$\beta = 0.05$ 57.6% (**-1.2%**)	$\beta = 0.2$ 57.8% (**-1.0%**)
`Loss Variants`	
+SFT Loss 60.0% (**+1.2%**)	KTO 59.8% (**+1.0%**)
`Data Construction Strategies`	
SENSE (Yang et al., 2024c) 58.4% (**-0.4%**)	

Table 7: Even with tricks applied, DPO still struggles to improve model performance on the Bird dataset (greedy decoding). The base model is Qwen2.5-7B-Instruct.

B Chain-of-Thought Solutions

In this section, we present the prompts used for Chain-of-Thought (CoT) reasoning synthesis and provide qualitative examples of the model's step-by-step Text-to-SQL responses.

B.1 Prompt for Synthesis

We carefully design the prompts used for CoT synthesis, as shown in Table 10. In our template, the gold SQL from the dataset is provided as a reference answer. This design enables the model to generate diverse reasoning paths during sampling while striving to maintain the correctness of the synthesized outputs.

B.2 Synthesized Chain-of-Thought

A synthesized Chain-of-Thought solution, generated by the `gpt-4o-mini-2024-07-18` on an instance from the Bird Train dataset, is illustrated in Table 11. The response begins with an analysis of the input question, followed by a step-by-step breakdown of the SQL generation process. After generating the SQL, the model further provides explanations for each component of the SQL query.

B.3 Samples From Syn CoT Models

We select a sample question from the Bird development set and compare the responses generated during the SFT stage and the DPO stage. The SFT-generated response is shown in Table 12, while the DPO-generated response is presented in the Table 13. Notably, DPO corrected an entity mismatch error present in the SFT response.

C Direct Preference Optimization

We provide a brief overview of direct preference optimization (DPO) and demonstrate how to utilize the trained DPO model to calculate the implicit reward of a response, as well as the credit assignment for each token within that response.

C.1 Learning Objective

For Reinforcement Learning with Human Feedback (RLHF) incorporating KL penalty, the learning objective is defined as (Ouyang et al., 2022):

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D}, y\sim\pi_\theta(y|x)} \left[r_\phi(x, y)\right] - \beta \mathbb{D}_{\text{KL}} \left[\pi_\theta(y \mid x) \,||\, \pi_{\text{ref}}(y \mid x)\right]$$

Here, π_{ref} and π_θ represent the initial model distribution and the optimized policy, respectively, while r_ϕ denotes a parameterized reward model.

Direct Preference Optimization (DPO) reformulates the objective by replacing the reward function with a differentiable form, reflecting the relationship between the optimal policy and the reward function. This leads to a new objective:

$$\mathbb{E}_{(x,y_w,y_l)\sim\mathcal{D}} \left[\log \sigma \left(\beta \log \frac{\pi_\theta(y_w \mid x)}{\pi_{\text{ref}}(y_w \mid x)} - \beta \log \frac{\pi_\theta(y_l \mid x)}{\pi_{\text{ref}}(y_l \mid x)}\right)\right]$$

In this formulation, y_w denotes the chosen output and y_l denotes the rejected output. The parameter β controls the penalty strength imposed by the KL divergence. By collecting pairwise preference data, the model can be optimized using supervised fine-tuning, achieving a performance comparable to RLHF (Rafailov et al., 2024b).

C.2 Implicit Reward

DPO implicitly encodes a reward model within the generative model. The reward of a given input-output pair (x, y) can be derived as:

$$r_\theta(x, y) = \beta \log \frac{\pi_\theta(y \mid x)}{\pi_{\text{ref}}(y \mid x)}$$

As the DPO training progresses, the optimized model simultaneously becomes a better generative model and a more refined reward model. After training, the implicit reward model, which is derived via conditional likelihood, can be independently used as a reward function (Lambert et al., 2024; Chen et al., 2024; Hosseini et al., 2024).

C.3 Token-level Credit Assignment

The implicit reward scores the entire output as a whole. By decomposing the conditional probability that featuring autoregressive generation process, the reward can be re-expressed as:

$$r_\theta(x, y) = \sum_{t=1}^{T} \beta \log \frac{\pi_\theta(y_t \mid x, y_{1:t-1})}{\pi_{\text{ref}}(y_t \mid x, y_{1:t-1})}$$

This decomposition allows for the calculation of token-level rewards, as the model score for each token can be identified separately. Although DPO training uses supervision at the full-sequence level, evidence has shown that the model can generalize compositionality to some extent, allowing it to distribute the reward signal to key tokens (Rafailov et al., 2024a). This facilitates credit assignment across the output sequence. The resulting dense reward can be utilized for further training and optimization (Zhong et al., 2024).

D Implementation Details

The computational environment used in our experiments is equipped with a 64-core Intel(R) Xeon(R) Platinum 8358 CPU @ 2.60GHz and 8 NVIDIA A800 GPUs with 80GB of memory each, running CUDA version 12.1.1.

Training is conducted using Llama Factory (Zheng et al., 2024), with FlashAttention 2.0 (Dao, 2023) and DeepSpeed ZeRO-3 (Rasley et al., 2020) enabled. For models larger than 10B parameters, the DPO stage utilizes CPU offloading for both model parameters and optimizer states. During inferenece, the trained models are hosted with vLLM (Kwon et al., 2023).

For training, we employ full-parameter fine-tuning with bf16 mixed-precision (Micikevicius et al., 2018). The optimizer used was AdamW (Loshchilov and Hutter, 2019) with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.99$). A cosine decay learning rate schedule and a linear warmup over the first 5% of training steps are also applied. The context window of models is set to 4096 tokens.

Across all training phases, we adopt a consistent batch size of 64. The learning rates for the SFT and DPO phases are set to 1×10^{-5} and 1×10^{-6}, respectively, for models smaller than 10B, and 7×10^{-6} and 7×10^{-7}, respectively, for models larger than 10B. The β parameter for DPO is set to 0.1.

All models are trained for 4 epochs during the SFT phase, and the best checkpoint is selected to serve as the reference model in DPO, based on the maj@K metric on the development set. Training is then continued for 8 epochs during the DPO phase.

Unless otherwise specified, during inference time across all stages (including chain-of-thought synthesis), the sampling budget is set to 16, with the $temperature$ and $topK$ parameters set to 1.0 and 32, respectively.

E Training Data Details

Quantity. In Bird dataset, the Vanilla SFT data consists of $9,428$ instances, while the CoT SFT data consists of $9,428 \times K$ instances (for Table 1, $K = 16$, as we generate 16 CoT solutions for each training sample in the original dataset).

The size of the DPO training data is smaller than that of the SFT training data and is model-dependent. This is because, during the construction of the preference data, we exclude data samples for which all SFT model-generated CoT solutions are either entirely correct or entirely incorrect. As a result, both Vanilla and Syn CoT preference datasets contain approximately 1.5k-2.5k preference pairs (e.g., Qwen-7b-Instruct Syn CoT has 1,546 pairs). The relationship between sample budgets and the quantity of DPO training data is illustrated in Figure 15 and Figure 16.

Construction of Input-Output Sequences and Their Average Length. Input prompt has an av-

erage length of 965 tokens, which is the same for Vanilla and Syn CoT settings. The input prompt includes not only the question but also database information, such as table and column names, primary and foreign key relationships, and potentially useful database values. Following CodeS (Li et al., 2024a), we first use a schema item classifier to identify the most relevant tables and columns based on the question. To improve recall accuracy, we replace the backbone model of the classifier from RoBERTa-Large (355M) (Liu, 2019) to XLM-RoBERTa-XL (3.5B) (Goyal et al., 2021). We then use the "coarse-to-fine" database value-matching approach to retrieve question-related values from the database. The retrieved tables, columns, values, and remaining primary and foreign keys form the database prompt.

The output label of Vanilla (gold SQL in the Bird dataset) has an average length of 44 tokens, while the synthesized chain-of-thought solutions have an average length of 404 tokens. Reported token numbers is measured by the tokenizer of Qwen-7b-Instruct.

F Data Quality Ablation

From Table 1, we can see that there are many cases where Syn CoT SFT has already surpassed Vanilla SFT, thus, it is natural to doubt that the benefit could be brought about by the potent proprietary model. In this section, we first analyze the ability of GPT-4o-mini on Text-to-SQL, then, we replace GPT-4o-mini with smaller open-sourced models to synthesize CoT reasoning paths, therefore further confirm that it is the chain-of-thought style solution path itself that enhance the effect of DPO.

Ability of the Synthesizers. We evaluate GPT-4o-mini's capability on Bird development set, as well as other open-sourced models that we will use as synthesizers. The result is shown in Table 8.

As a comparison, Qwen2.5-7B-Instruct, after fine-tuning on original Bird Train data, reaches 58.8, which is higher than the EX of any synthesizer, as shown in Table 1.

It is evident that directly prompting closed-source instruction-tuned models does not inherently offer an advantage in the Text-to-SQL task, consistent with findings from previous work (Li et al., 2024a). Closed-source models only perform well when incorporating with complex multi-agent designs (Pourreza et al., 2024). Therefore, in this paper, we use GPT-4o-mini merely as a substitute for manual effort, allowing us to quickly and cost-effectively obtain CoT solutions of Text-to-SQL data, rather than distilling capabilities from a powerful model.

Model	Bird Dev		
	ZS-CoT	FS-SQL	FS-CoT
GPT-4o-mini	50.1	55.3	53.0
Qwen(1.5B)	15.6	23.1	18.9
Qwen(7B)	41.8	47.6	43.1
Qwen(32B)	52.9	55.3	53.0

Table 8: Performance of models used for chain-of-thought solution synthesis under different prompting strategies (Greddy). **Qwen (XB)**: Qwen2.5-XB-Instruct; **ZS-CoT**: Zero-shot CoT; **FS-SQL**: 3-Shot examples randomly chosen from Bird Train; **FS-CoT**: 3-Shot examples randomly chosen from its own synthesized CoT solutions on Bird Train.

DPO with Different CoT Quailities. As shown in Table 8, CoT quality varies for different size of synthesizers. We then use each of them to synthesize CoT solutions, and train the corresponding Syn CoT model separately (Base model is Qwen2.5-7B-Instruct). The results are presented in Table 9.

Synthesizer	Bird Dev	
	SFT	DPO
GPT-4o-mini	57.4	61.9 (+4.5)
Qwen(1.5B)	40.9	59.1 (+18.2)
Qwen(7B)	54.3	60.6 (+6.3)
Qwen(32B)	59.3	62.5 (+3.2)

Table 9: Performance of models trained from CoT-enhanced data generated by different synthesizers (Greddy). **Qwen (XB)**: Qwen2.5-XB-Instruct.

Its evident that all these models achieve significant improvement in the DPO phase. Interestingly, the impact of CoT quality on model performance is significantly weakened after DPO.

G Results on Spider and Its Robustness Variants

In addition to the Bird benchmark, we also train and evaluate three representative base models on the Spider benchmark. We further assess the models trained on Spider with robustness test sets.

However, during our experiments, we identified several issues with using CoT and DPO techniques on the Spider dataset:

Simplicity of SQL in Spider. Most SQL queries in Spider are quite simple. A typical example is: `Q: How many concerts are there in year 2014 or 2015? A: SELECT count(*) FROM concert WHERE year = 2014 OR year = 2015.` smaller pre-trained language model methods have already achieved good results on these queries (Li et al., 2023). Additionally, the best models on the Spider benchmark have reached human-level performance (91.2%[2]). It is widely acknowledged that CoT often does not perform better on simple questions, possibly due to overthinking issues (Sprague et al., 2024).

Inaccurate Feedback on Execution Results in Spider. When constructing preference data, we rely on execution results on the database to judge the correctness of sampled SQL. The Bird dataset's databases are specially designed with massive rows, whereas Spider's databases have fewer rows (#Row/DB: Bird ($549K$) vs. Spider ($2K$), $< 0.5\%$ (Li et al., 2024b)), leading to potential false positives during evaluation (Zhong et al., 2020). Although subsequent work has attempted to mitigate this by constructing multiple test suite (TS) databases, the Spider benchmark only provides TS databases for the development set, not for the training set, leading to inaccurate preference pair construction.

Small Scale of Constructible Preference Data on Spider. Due to the first issue, most models on the Spider dataset achieve very high accuracy on the training set (Pass@16 $\geq 99\%$, compared to Bird's $\leq 80\%$), resulting in a very small preference dataset ($0.1 - 0.3k$, with the SFT phase dataset being $7k$. As for comparison, DPO data on Bird is approximately $1.5 - 2.5k$).

Based on these issues, we choose to focus primarily on the model's performance on the Bird dataset, one of the most challenging Text-to-SQL benchmarks that closely reflects real-world scenarios. Therefore, results related to Spider are included as a reference in the appendix due to constraints of limited space, as shown below.

G.1 Spider

The model performance on Spider's development set is summarized in Table 14. Despite the aforementioned challenges, the Syn-CoT model consistently achieved improvements during the DPO stage.

[2]from the Spider Leaderboard: ttps://yale-lily.github.io/spider

G.2 Spider Variants

We select the best checkpoint according to the Spider development set and directly evaluate it on these robustness test sets. The results for Spider-Syn, Spider-Realistic, and Spider-DK are presented in Table 15. The CoT model continues to demonstrate consistent performance improvements during the DPO stage, further confirming its strong generalization capabilities after DPO training.

Dr. Spider is a more comprehensive and sophisticated robustness test set, which categorizes all perturbations into three major types: database (DB), natural language question (NLQ), and SQL query. It then further subdivides them into 17 subcategories. For each type of perturbation, dedicated test sets are constructed (Chang et al., 2023).

For DB perturbations, the results of Syn CoT models are shown in Table 16, alongside the results of the vanilla model in Table 17. For NLQ perturbations, the results of Syn CoT models are shown in Table 18, alongside the results of the vanilla model in Table 19. For SQL perturbations, the results of Syn CoT models are shown in Table 20, alongside the results of the vanilla model in Table 21.

Except for a few specific cases, Syn CoT model still demonstrates consistent improvements during the DPO stage.

H Error Classifications

H.1 Description

In this paper, we classify errors made by predicted SQL into 6 major categories with 17 variant types. Descriptions of each category or type are shown in Table 22.

H.2 Error Samples

We provide samples for each error type in our classification criteria, for external knowledge, see Table 23, for schema linking, see Table 24, for value retrieval, see Table 25, for operation, see Table 26, for information, see Table 27, for syntax error, see Table 28. These are selected from model predictions on the Bird development set.

H.3 Classification Result

The total number of errors and the proportion of each error type are presented in pie charts. Results of Syn CoT models is shown in Figure 7, and results of vanilla models is shown in Figure 8.

A SQL query may commit multiple types of errors simultaneously. In our analysis, however, we only attribute each erroneous SQL to the most prominent type of mistake it made logically, for the convenience of analysis.

I More Analysis of DPO

I.1 Overall Effect

The changes in the correctness of model-generated outputs before and after the DPO stage are illustrated in Figure 9. The CoT model corrects a greater number of errors during the DPO stage, while the proportion of newly introduced errors remained comparable to that of the vanilla model.

I.2 Effect on Difficulty Classes

We analyze the impact of DPO on questions of varying difficulty levels, as shown in Figure 10. The CoT model exhibits significant improvements in performance on medium- and high-difficulty questions during the DPO stage, while the improvements on simpler questions are relatively limited.

I.3 Fix Rate Difference

To facilitate a detailed comparison of how the CoT model enhances DPO's error correction capabilities for different error types, we rank the error types by the increase in fix rates introduced by DPO. The results are presented in Table 29.

I.4 Emerging Errors

The number of newly introduced errors during the DPO process is summarized in Table 30. Overall, the distribution of emerging errors in the Syn CoT model is similar to that of the vanilla model. However, while CoT improves DPO's ability to address hallucination errors, it also leads to an increase in newly introduced hallucinations during the DPO stage.

I.5 Transition Matrices

The transitions between error categories before and after DPO are depicted in Figure 11. For a more detailed view of these transitions, the vanilla model's error type transitions are shown in Figure 12, while those of the Syn CoT model are presented in Figure 13.

I.6 Weakness

Existing Text-to-SQL Methods Can Complement DPO's Weaknesses. Despite improvements brought by CoT, DPO remains less effective at fixing certain error types. However, these align with core challenges that existing Text-to-SQL methods aim to address. For example, the model frequently fails to recall relevant Table (Fix 15.9%) and Column (Fix 16.1%), a key challenge of schema linking. Notable recent works include CHESS (Talaei et al., 2024) and E-SQL (Caferoglu and Ulusoy, 2024). Syntax Error (Fix 13.3%) is another tricky problem. Wang et al. (2024b); Askari et al. (2024) have proposed post-generation execution and repair strategies to ensure executable returned SQL.

J Experiment Design

J.1 Evaluation Preference Dataset

To ensure a fair comparison of the discriminative capabilities between the Vanilla and CoT models, we construct the mentioned evaluation preference dataset as follows.

First, both SFT models are used to sample from the development set. For any data point where both models could generate paired data ($i.e.$, both could simultaneously sample a positive and a negative example), we use the sampling outcomes from the CoT model to randomly construct a preference pair. Subsequently, we extract the SQL portion of the pair to serve as the preference pair for the vanilla model, incorporating it into their respective evaluation sets. Through this construction process, we ensure that the databases, questions, and SQLs in the dataset for both models are identical.

K Prominent Reward Hacking Patterns

In this section, we provide other prominent reward hacking patterns of the DPO training process in the Vanilla setting from our observations, as illustrated in Table 31, 32, 33.

L More Scaling Results

Scaling behavior of performance on the CoT synthesis budget and sample budget of preference data collection under all inference strategies are complemented in Figure 14 and Figure 15, respectively.

It is noteworthy that performance saturation regarding sample budget in the preference data collection stage is mainly caused by the diminishing return of new preference pairs, as can be clearly seen from the log-scale plot Figure 16.

M Application Details

M.1 DTS-SQL

DTS-SQL divides the Text-to-SQL task into two stages: Schema-Linking and SQL-Generation (Pourreza and Rafiei, 2024b). Based on the code available in its repository, we construct the training and testing datasets for the SQL-Generation stage. (Since the original code is developed for the Spider dataset, we refer to the submitted `DTS_SQL_BIRD_submission.py` file to account for incorporating Hints when constructing Prompts for the Bird dataset.) At this stage, we obtain a dataset without CoT. Subsequently, we utilize `gpt-4o-mini-2024-07-18` to generate 4 CoT paths, thereby creating two distinct datasets for training our two types of models.

M.2 C3

C3 stands for Clear Prompting, Calibration of Model Bias, and Consistency Output (Dong et al., 2023). For the Clear Prompting and Calibration of Model Bias components, we use the C3 prompt templates, removing parts related to CoT. We run the Schema-Linking code to generate prompt inputs, which are paired with the correct SQL to form input-output pairs. Additionally, we use `gpt-4o-mini-2024-07-18` to generate 4 CoT paths, thereby creating two datasets for training our two models. After training the models, we test on Spider Dev using a Consistency Output subset of size 16, identical to Maj@16.

System

You are a senior data analyst who expertise at structural query language (SQL). Given a question made by front end employees and targeted database schema, you are asked to translate that question into SQLite query with detailed explanation.

Additionally, the input will be accompanied with a reference solution from your colleagues, which may or may not be correct. This extra information intents to help you to formulate your answer, and you are asked not to mention reference solution in any form.

To facilitate SQL extraction with regular expression, the SQL in your answer should be expressed in a Markdown code block with proper highlight. For example,
```SQL
SELECT * FROM database;
```

Table 10: The system message we use when prompting `gpt-4o-mini-2024-07-18` to synthesize Chain-of-Thought reasoning solutions for publicly available Text-to-SQL datasets.

Response

Database Prompt: ...
Question: Among the payments made by Mary Smith, how many of them are over 4.99?
External Knowledge: over 4.99 refers to amount > 4.99
Gold SQL: SELECT count(payment.amount) FROM payment INNER JOIN customer ON payment.customer_id = customer.customer_id WHERE customer.first_name = 'MARY' AND customer.last_name = 'SMITH' AND payment.amount > 4.99

Synthesized Chain-of-Thought Solution
To answer the question about how many payments made by Mary Smith are greater than 4.99, we will need to join the `payment` table with the `customer` table. The query will count the number of payments that match Mary Smith's first and last name while ensuring the payment amount exceeds 4.99.

Here's how the SQL query can be constructed:
1. **Join the Tables**: We will perform INNER JOIN between the `payment` and `customer` tables on the `customer_id` column. This links payments to the customers who made them.
2. **Filter Conditions**: We will add conditions to filter for only those rows where the customer's first name is 'MARY' and last name is 'SMITH'.
3. **Amount Condition**: Additionally, we will include a condition to check if the payment amount is greater than 4.99.
4. **Count the Results**: Finally, we will use the COUNT function to return the total number of matching records.

Here is the SQL query that fulfills the requirements:
```SQL
SELECT COUNT(payment.amount)
FROM payment
INNER JOIN customer ON payment.customer_id = customer.customer_id
WHERE customer.first_name = 'MARY'
AND customer.last_name = 'SMITH'
AND payment.amount > 4.99;
```

Explanation:
- `SELECT COUNT(payment.amount)`: This part specifies that we want to count the number of payments that meet the conditions specified.
- `FROM payment`: Indicates that the main table we are querying is the `payment` table.
- `INNER JOIN customer ON payment.customer_id = customer.customer_id`: This joins the `payment` table with the `customer` table based on the `customer_id` present in both tables, allowing us to access customer details related to each payment.
- `WHERE customer.first_name = 'MARY' AND customer.last_name = 'SMITH'`: These conditions filter the results to include only payments made by a customer named Mary Smith.
- `AND payment.amount > 4.99`: This additional condition specifies that we only want payments where the amount is greater than 4.99.

This query will count the payments made by Mary Smith that exceed the specified amount.

Table 11: A qualitative example of `gpt-4o-mini-2024-07-18` generated Chain-of-Thought for an instance of Bird Train set. **SQL in the CoT can be different from original gold SQL in the dataset**.

Response

Database Prompt: Table district, columns = [district.a11 (integer | comment : average salary | values : 12541 , 8507) , district.a3 (text | comment : region | values : Prague , central Bohemia) , district.district_id (integer | primary key | comment : location of branch | values : 1 , 2) ...
Question: How many male customers who are living in North Bohemia have average salary greater than 8000?
External Knowledge: Male means that gender = 'M'; a3 refers to region; a11 pertains to average salary.

To answer the question of how many male customers living in North Bohemia have an average salary greater than 8000, we need to join the `district` and `client` tables based on the `district_id`. We will then filter the results based on the specified conditions: the region being 'North Bohemia', the gender being 'M', and the average salary being greater than 8000.

Here's the SQL query that accomplishes this:

```SQL
SELECT COUNT(client.client_id)
FROM district
INNER JOIN client ON district.district_id = client.district_id
WHERE district.a3 = 'North Bohemia'
AND client.gender = 'M'
AND district.a11 > 8000;
```

Explanation:
1. **SELECT COUNT(client.client_id)**: This part of the query counts the number of male clients that meet the specified criteria.
2. **FROM district**: We start our query from the `district` table since we need to filter based on the region and average salary.
3. **INNER JOIN client ON district.district_id = client.district_id**: We perform an inner join between the `district` and `client` tables using the `district_id` as the key. This allows us to access the client information associated with each district.
4. **WHERE district.a3 = 'North Bohemia'**: This condition filters the results to only include districts in the 'North Bohemia' region.
5. **AND client.gender = 'M'**: This condition filters the results to only include male clients.
6. **AND district.a11 > 8000**: This condition filters the results to only include districts where the average salary is greater than 8000.

By executing this query, we will obtain the count of male customers living in North Bohemia with an average salary greater than 8000.

Table 12: The full response Syn CoT SFT Qwen2.5-7B-Instruct generated for a sample question from Bird Dev. **The response is wrong due to entity mismatch for the region 'north Bohemia'**.

Response

Database Prompt: Table district, columns = [district.a11 (integer | comment : average salary | values : 12541 , 8507) , district.a3 (text | comment : region | values : Prague , central Bohemia) , district.district_id (integer | primary key | comment : location of branch | values : 1 , 2) ...
Question: How many male customers who are living in North Bohemia have average salary greater than 8000?
External Knowledge: Male means that gender = 'M'; a3 refers to region; a11 pertains to average salary.

To answer the question of how many male customers living in North Bohemia have an average salary greater than 8000, we need to join the `client` table with the `district` table based on the `district_id`. We will then apply the necessary filters for the region (`a3`), gender (`gender`), and average salary (`a11`).

Here's the breakdown of the SQL query:

1. **SELECT count(client.client_id)**: We want to count the number of male clients, so we use the 'count' function on 'client.client_id'.
2. **FROM district INNER JOIN client ON district.district_id = client.district_id**: We perform an inner join between the `district` and `client` tables on the `district_id` column, which is common to both tables. This allows us to access the district information for each client.
3. **WHERE district.a3 = 'north Bohemia'**: We filter the results to include only those clients who live in the North Bohemia region.
4. **AND client.gender = 'M'**: We further filter to include only male clients.
5. **AND district.a11 > 8000**: Finally, we add a condition to ensure that we only count clients whose average salary is greater than 8000.

Putting this all together, the SQL query is as follows:

```SQL
SELECT count(client.client_id)
FROM district
INNER JOIN client ON district.district_id = client.district_id
WHERE district.a3 = 'north Bohemia'
AND client.gender = 'M'
AND district.a11 > 8000;
```

This query will return the count of male clients residing in North Bohemia with an average salary exceeding 8000.

Table 13: The full response Syn CoT DPO Qwen2.5-7B-Instruct generated for a sample question from Bird Dev. **The model is able to infer entity format of 'north Bohemia' from value examples given in database prompt**.

	Model	Spider Dev						Δ
		Greedy		Pass@1		Maj@16		
		SFT	DPO	SFT	DPO	SFT	DPO	
Execution Accuracy (EX)								
Vanilla	Deepseek-coder-6.7b-instruct	81.6	81.0 (-0.6)	80.0	80.4 (+0.4)	82.5	81.9 (-0.6)	-
	Qwen2.5-7B-Instruct	79.8	79.7 (-0.1)	78.4	78.6 (+0.2)	81.8	81.6 (-0.2)	-
	CodeS-7b	79.3	79.4 (+0.1)	78.0	78.2 (+0.2)	80.9	80.4 (-0.5)	-
Syn CoT	Deepseek-coder-6.7b-instruct	80.0	82.1 (+2.1)	79.4	81.3 (+1.9)	82.8	83.8 (+1.0)	82.5 → 83.8 (**+1.3**)
	Qwen2.5-7B-Instruct	79.1	80.5 (+1.4)	76.8	78.9 (+2.1)	80.4	82.6 (+2.2)	81.8 → 82.6 (**+0.8**)
	CodeS-7b	76.7	80.0 (+3.3)	75.5	78.8 (+3.3)	78.3	82.3 (+4.0)	80.9 → 82.3 (**+1.4**)
Test Suite (TS)								
Vanilla	Deepseek-coder-6.7b-instruct	80.5	80.7 (+0.2)	79.1	79.6 (+0.5)	81.1	80.7 (-0.4)	-
	Qwen2.5-7B-Instruct	78.5	78.9 (+0.4)	77.2	77.1 (+0.1)	79.5	79.3 (-0.2)	-
	CodeS-7b	76.9	77.1 (+0.2)	75.4	75.7 (+0.3)	78.1	77.7 (-0.4)	-
Syn CoT	Deepseek-coder-6.7b-instruct	78.3	80.3 (+2.0)	76.9	78.9 (+2.0)	79.7	81.6 (+1.9)	81.1 → 81.6 (**+0.5**)
	Qwen2.5-7B-Instruct	77.5	77.9 (+0.4)	75.0	76.6 (+1.6)	78.6	80.2 (+1.6)	79.5 → 80.2 (**+0.7**)
	CodeS-7b	74.8	77.0 (+2.2)	73.1	75.3 (+2.2)	76.1	77.8 (+1.7)	78.1 → 77.8 (**-0.3**)

Table 14: Model performance on the Spider development set. **Vanilla**: SFT and DPO on the original Spider training set; **Syn CoT**: SFT and DPO on the CoT-enhanced training set; Δ: The performance difference in EX/TS between "Syn CoT + DPO" and "Vanilla + SFT" when using the same base model.**We make preference dataset with EX since TS in not available to train set.**

	Model	Spider Variants Dev (EX)						ΔEX
		Greedy		Pass@1		Maj@16		
		SFT	DPO	SFT	DPO	SFT	DPO	
Spider-Syn								
Vanilla	Deepseek-coder-6.7b-instruct	73.8	73.5 (-0.3)	72.5	72.4 (-0.1)	75.2	75.5 (+0.3)	-
	Qwen2.5-7B-Instruct	71.5	72.0 (+0.5)	69.6	69.8 (+0.2)	73.7	73.6 (-0.1)	-
	CodeS-7b	69.1	69.4 (+0.3)	67.1	67.6 (+0.5)	71.1	71.2 (+0.1)	-
Syn CoT	Deepseek-coder-6.7b-instruct	70.7	71.7 (+1.0)	70.1	71.3 (+1.2)	76.7	76.8 (+0.1)	75.2 → 76.8 (**+1.6**)
	Qwen2.5-7B-Instruct	69.7	71.2 (+1.5)	67.9	70.5 (+2.6)	74.1	76.2 (+2.1)	73.7 → 76.2 (**+2.5**)
	CodeS-7b	64.4	69.1 (+4.7)	63.6	67.5 (+3.9)	69.8	71.4 (+1.6)	71.1 → 71.4 (**+0.3**)
Spider-Realistic								
Vanilla	Deepseek-coder-6.7b-instruct	77.2	77.2	77.0	76.5 (-0.5)	78.9	78.3 (-0.6)	-
	Qwen2.5-7B-Instruct	75.4	75.4	73.7	74.2 (+0.5)	76.6	77.4 (+0.8)	-
	CodeS-7b	73.8	74.0 (+0.2)	73.0	72.9 (-0.1)	76.4	75.8 (-0.6)	-
Syn CoT	Deepseek-coder-6.7b-instruct	79.7	80.7 (+1.0)	77.4	78.7 (+1.3)	80.9	82.7 (+1.8)	78.9 → 82.7 (**+3.8**)
	Qwen2.5-7B-Instruct	76.0	77.4 (+1.4)	73.9	75.1 (+1.2)	78.3	79.1 (+0.8)	76.6 → 79.1 (**+2.5**)
	CodeS-7b	73.8	76.4 (+2.6)	71.5	73.3 (+1.8)	76.6	78.5 (+1.9)	76.4 → 78.5 (**+2.1**)
Spider-DK								
Vanilla	Deepseek-coder-6.7b-instruct	69.0	69.8 (-0.2)	68.2	68.5 (+0.3)	70.3	70.1 (-0.2)	-
	Qwen2.5-7B-Instruct	70.5	70.3 (-0.2)	69.0	69.4 (+0.4)	73.8	73.6 (-0.2)	-
	CodeS-7b	67.5	67.5	66.3	66.6 (+0.3)	69.2	68.6 (-0.6)	-
Syn CoT	Deepseek-coder-6.7b-instruct	69.7	72.1 (+2.4)	68.4	71.2 (+2.8)	72.9	75.1 (+2.2)	70.3 → 75.1 (**+4.8**)
	Qwen2.5-7B-Instruct	67.5	69.0 (+1.5)	65.1	67.2 (+2.1)	70.7	72.9 (+2.2)	73.8 → 72.9 (**-0.9**)
	CodeS-7b	62.6	67.7 (+5.1)	61.9	67.0 (+5.1)	67.5	72.1 (+4.6)	69.2 → 72.1 (**+2.9**)

Table 15: Model performance on Spider's variants (Spider-Syn, Spider-Realistiv, Spider-DK). **Vanilla**: SFT and DPO on the original Spider training set; **Syn CoT**: SFT and DPO on the CoT-enhanced training set; **ΔEX**: The performance difference in EX between "Syn CoT + DPO" and "Vanilla + SFT" when using the same base model. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Syn CoT Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
DB Perturbations							
schema-synonym	Greedy	66.6	68.8 (+2.2)	63.1	65.1 (+2.0)	57.5	62.2 (+4.7)
	Pass@1	63.8	66.7 (+2.9)	60.7	63.3 (+2.6)	55.0	60.8 (+5.8)
	Maj@K	69.7	71.7 (+2.0)	68.1	69.3 (+1.2)	63.7	66.2 (+2.5)
schema-abbreviation	Greedy	74.8	77.9 (+3.1)	70.7	73.3 (+2.6)	69.4	74.9 (+5.5)
	Pass@1	73.0	76.1 (+3.1)	68.1	71.2 (+3.1)	66.8	73.2 (+6.4)
	Maj@K	76.7	79.8 (+3.1)	75.3	77.6 (+2.3)	73.6	77.1 (+3.5)
DBcontent-equivalence	Greedy	61.5	63.1 (+1.6)	52.1	50.8 (-1.3)	56.0	59.2 (+3.2)
	Pass@1	59.5	60.3 (+0.8)	49.2	49.4 (+0.2)	56.4	56.8 (+0.4)
	Maj@K	61.8	63.1 (+1.3)	55.5	55.0 (-0.5)	61.5	62.6 (+1.1)
Average	Greedy	67.6	69.9 (+2.3)	62.0	63.1 (+1.1)	61.0	65.4 (+4.4)
	Pass@1	65.4	67.7 (+2.3)	59.3	61.3 (+2.0)	59.4	63.6 (+4.2)
	Maj@K	69.4	71.5 (+2.1)	66.3	67.3 (+1.0)	66.3	68.6 (+2.3)

Table 16: Syn CoT model performance on DB perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Vanilla Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
DB Perturbations							
schema-synonym	Greedy	66.3	66.4 (+0.1)	63.3	63.5 (+0.2)	62.0	62.2 (+0.2)
	Pass@1	65.6	65.6 (+0.0)	61.1	61.2 (+0.1)	59.7	60.5 (+0.8)
	Maj@K	69.0	68.6 (-0.4)	67.9	67.8 (-0.1)	65.7	65.9 (+0.2)
schema-abbreviation	Greedy	77.0	77.3 (+0.3)	75.3	75.7 (+0.4)	71.3	71.3
	Pass@1	75.4	75.4	73.3	73.5 (+0.2)	69.8	70.1 (+0.3)
	Maj@K	78.2	77.9 (-0.3)	77.1	77.7 (+0.6)	74.2	74.2
DBcontent-equivalence	Greedy	52.9	52.4 (-0.5)	63.4	63.4	62.0	62.3 (+0.3)
	Pass@1	54.1	53.9 (-0.2)	61.6	61.6	61.0	61.0
	Maj@K	53.4	53.4	69.1	69.1	64.4	63.6 (-0.8)
Average	Greedy	65.4	65.4	67.3	67.5 (+0.2)	65.1	65.3 (+0.2)
	Pass@1	65.0	65.0	65.3	65.4 (+0.1)	63.5	63.9 (+0.4)
	Maj@K	66.9	66.6 (-0.3)	71.4	71.5 (+0.1)	68.1	67.9 (-0.2)

Table 17: Vanilla model performance on DB perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Syn CoT Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
NLQ Perturbations							
keyword-synonym	Greedy	67.4	71.2 (+3.8)	66.7	68.9 (+2.2)	62.5	71.9 (+9.4)
	Pass@1	66.9	70.4 (+3.5)	65.6	68.5 (+2.9)	60.6	68.8 (+8.2)
	Maj@K	69.9	73.5 (+3.6)	69.7	71.5 (+1.8)	66.7	72.8 (+6.1)
keyword-carrier	Greedy	76.4	80.7 (+4.3)	77.2	78.9 (+1.7)	77.4	84.5 (+7.1)
	Pass@1	78.8	81.6 (+2.8)	74.9	76.3 (+1.4)	77.1	83.2 (+6.1)
	Maj@K	79.9	83.7 (+3.8)	76.7	78.9 (+2.2)	78.9	85.7 (+6.8)
column-synonym	Greedy	57.9	59.9 (+2.0)	54.2	55.6 (+1.4)	52.0	53.3 (+1.3)
	Pass@1	54.5	57.1 (+2.6)	52.7	54.0 (+1.3)	49.2	52.4 (+3.2)
	Maj@K	59.0	60.6 (+1.6)	57.7	58.3 (+0.6)	54.4	58.1 (+3.7)
column-carrier	Greedy	67.5	71.8 (+4.3)	70.6	71.2 (+0.6)	68.6	73.7 (+5.1)
	Pass@1	67.2	72.5 (+5.3)	68.2	70.2 (+2.0)	65.9	72.8 (+6.9)
	Maj@K	74.3	75.8 (+1.5)	72.7	74.4 (+1.7)	73.9	77.7 (+3.8)
column-attribute	Greedy	67.2	68.1 (+0.9)	57.1	58.0 (+0.9)	52.1	56.3 (+4.2)
	Pass@1	60.7	64.0 (+3.3)	55.7	57.8 (+2.1)	49.1	53.4 (+4.3)
	Maj@K	70.6	71.4 (+0.8)	61.3	63.0 (+1.7)	58.0	59.7 (+1.7)
column-value	Greedy	76.0	76.0	75.7	76.6 (+0.9)	74.3	77.6 (+3.3)
	Pass@1	73.5	75.1 (+1.6)	72.8	73.7 (+0.9)	70.9	74.1 (+3.2)
	Maj@K	77.3	78.0 (+0.7)	77.3	78.0 (+0.7)	75.3	78.9 (+3.6)
value-synonym	Greedy	67.2	68.0 (+0.8)	63.6	64.4 (+0.8)	64.2	65.2 (+1.0)
	Pass@1	64.6	66.0 (+1.4)	61.6	63.3 (+1.7)	60.8	63.5 (+2.7)
	Maj@K	68.6	69.4 (+0.8)	65.0	67.0 (+2.0)	66.6	67.2 (+0.6)
multitype	Greedy	63.6	66.5 (+2.9)	61.0	63.5 (+2.5)	60.8	63.9 (+3.1)
	Pass@1	61.6	65.1 (+3.5)	60.5	62.4 (+1.9)	57.8	62.0 (+4.2)
	Maj@K	66.5	68.5 (+2.0)	64.9	66.8 (+1.9)	64.4	67.1 (+2.7)
others	Greedy	73.2	75.5 (+2.3)	73.5	75.3 (+1.8)	70.6	74.9 (+4.3)
	Pass@1	72.8	75.1 (+2.3)	71.7	73.3 (+1.6)	68.9	73.3 (+4.4)
	Maj@K	76.2	77.7 (+1.5)	76.1	77.0 (+0.9)	73.4	76.6 (+3.2)
Average	Greedy	67.6	69.9 (+2.3)	62.0	63.1 (+1.1)	61.0	65.4 (+4.4)
	Pass@1	65.4	67.7 (+2.3)	59.3	61.3 (+2.0)	59.4	63.6 (+4.2)
	Maj@K	69.4	71.5 (+2.1)	66.3	67.3 (+1.0)	66.3	68.6 (+2.3)

Table 18: Syn CoT model performance on NLQ perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Vanilla Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
NLQ Perturbations							
keyword-synonym	Greedy	69.6	69.8 (+0.2)	68.2	68.3 (+0.1)	66.8	67.7 (+0.9)
	Pass@1	68.7	68.8 (+0.1)	67.1	67.0 (-0.1)	65.3	66.0 (+0.7)
	Maj@K	70.9	70.4 (-0.5)	69.4	69.5 (+0.1)	68.9	68.5 (-0.4)
keyword-carrier	Greedy	80.7	81.5 (+0.8)	80.2	80.5 (+0.3)	81.7	81.2 (-0.5)
	Pass@1	80.2	80.3 (+0.1)	79.0	79.2 (+0.2)	79.8	79.9 (+0.1)
	Maj@K	81.0	81.0	80.7	79.9 (-0.8)	83.0	84.0 (+1.0)
column-synonym	Greedy	59.9	60.0 (+0.1)	57.2	57.4 (+0.2)	54.9	55.1 (+0.2)
	Pass@1	59.6	59.4 (-0.2)	55.7	56.0 (+0.3)	54.2	54.3 (+0.1)
	Maj@K	62.0	61.6 (-0.4)	59.0	59.3 (+0.3)	56.8	57.0 (+0.2)
column-carrier	Greedy	68.4	68.2 (-0.2)	74.6	74.3 (-0.3)	77.2	77.4 (+0.2)
	Pass@1	68.5	68.6 (+0.1)	72.2	72.5 (+0.3)	75.0	74.9 (-0.1)
	Maj@K	70.1	69.8 (-0.3)	76.7	76.2 (-0.5)	78.4	77.9 (-0.5)
column-attribute	Greedy	59.7	59.7	49.6	50.4 (+0.8)	47.9	47.9
	Pass@1	58.6	58.7 (+0.1)	49.4	49.3 (-0.1)	46.7	47.4 (+0.7)
	Maj@K	64.7	65.5 (+0.8)	53.8	53.8	51.3	54.6 (+3.3)
column-value	Greedy	74.7	74.7	76.6	76.6	71.4	71.7 (+0.3)
	Pass@1	74.2	74.1 (-0.1)	73.3	73.5 (+0.2)	70.1	70.8 (+0.7)
	Maj@K	75.7	75.7	76.6	76.0 (-0.6)	75.3	74.3 (-1.0)
value-synonym	Greedy	67.0	66.8 (-0.2)	65.2	65.6 (+0.4)	64.2	64.0 (-0.2)
	Pass@1	66.1	66.4 (+0.3)	63.4	63.9 (+0.5)	62.9	63.3 (+0.4)
	Maj@K	70.0	69.2 (-0.8)	66.2	66.4 (+0.2)	66.0	65.4 (-0.6)
multitype	Greedy	63.7	63.7	62.0	62.2 (+0.2)	63.1	63.3 (+0.2)
	Pass@1	62.9	63.2 (+0.3)	60.7	61.0 (+0.3)	61.2	61.5 (+0.3)
	Maj@K	65.1	65.1	64.0	64.2 (+0.2)	64.1	64.7 (+0.6)
others	Greedy	75.6	75.5 (-0.1)	75.4	75.4	73.6	73.7 (+0.1)
	Pass@1	74.9	75.0 (+0.1)	73.9	74.1 (+0.2)	72.7	72.8 (+0.1)
	Maj@K	76.7	76.8 (+0.1)	76.4	76.3 (-0.1)	75.0	74.6 (-0.4)
Average	Greedy	68.8	68.9 (+0.1)	67.7	67.9 (+0.2)	66.8	66.9 (+0.1)
	Pass@1	68.2	68.3 (+0.1)	66.1	66.3 (+0.2)	65.3	65.7 (+0.4)
	Maj@K	70.7	70.6 (-0.1)	69.2	69.1 (-0.1)	68.8	69.0 (+0.2)

Table 19: Vanilla model performance on NLQ perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Syn CoT Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
SQL Perturbations							
comparison	Greedy	68.0	74.2 (+6.2)	57.9	64.6 (+6.7)	68.0	75.3 (+7.3)
	Pass@1	68.0	73.5 (+5.5)	60.8	66.5 (+5.7)	66.0	74.2 (+8.2)
	Maj@K	75.3	77.5 (+2.2)	62.9	70.2 (+7.3)	76.4	78.7 (+2.3)
sort-order	Greedy	71.4	78.6 (+7.2)	69.3	73.4 (+4.1)	63.0	74.5 (+11.5)
	Pass@1	69.2	75.6 (+6.4)	69.3	74.7 (+5.4)	64.9	73.7 (+8.8)
	Maj@K	73.4	78.1 (+4.7)	72.4	78.1 (+5.7)	69.3	76.6 (+7.3)
nonDB-number	Greedy	76.3	78.6 (+2.3)	76.3	77.1 (+0.8)	77.1	80.2 (+3.1)
	Pass@1	76.6	77.3 (+0.7)	75.2	76.4 (+1.2)	76.3	76.1 (-0.2)
	Maj@K	80.2	80.1 (-0.1)	79.4	80.9 (+1.5)	81.7	82.4 (+0.7)
DB-text	Greedy	77.5	78.8 (+1.3)	75.1	75.7 (+0.6)	72.6	75.0 (+2.4)
	Pass@1	77.0	77.5 (+0.5)	72.8	74.3 (+1.5)	69.4	72.4 (+3.0)
	Maj@K	81.9	81.8 (-0.1)	79.4	79.6 (+0.2)	74.5	76.0 (+1.5)
DB-number	Greedy	83.9	83.9	80.5	82.7 (+2.2)	81.0	81.5 (+0.5)
	Pass@1	79.6	81.3 (+1.7)	78.9	80.1 (+1.2)	75.3	76.9 (+1.6)
	Maj@K	83.4	84.9 (+1.5)	82.9	83.4 (+0.5)	79.8	80.2 (+0.4)
Average	Greedy	75.4	78.8 (+3.4)	71.8	74.7 (+2.9)	72.3	77.3 (+5.0)
	Pass@1	74.1	77.0 (+2.9)	71.4	74.4 (+3.0)	70.4	74.7 (+4.3)
	Maj@K	78.8	80.5 (+1.7)	75.4	78.4 (+3.0)	76.3	78.8 (+2.5)

Table 20: Syn CoT model performance on SQL perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Type	Metrics	Vanilla Models					
		DSC (6.7B)		Qwen (7B)		CodeS (7B)	
		SFT	DPO	SFT	DPO	SFT	DPO
SQL Perturbations							
comparison	Greedy	70.2	69.7 (-0.5)	69.1	69.1	71.9	72.5 (+0.6)
	Pass@1	69.7	69.6 (-0.1)	67.7	68.3 (+0.6)	68.9	69.5 (+0.6)
	Maj@K	73.6	73.6	71.9	71.9	72.5	73.0 (+0.5)
sort-order	Greedy	79.2	79.2	75.5	76.0 (+0.5)	78.6	77.6 (-1.0)
	Pass@1	77.4	77.4	74.0	73.9 (-0.1)	74.5	74.3 (-0.2)
	Maj@K	80.7	80.2 (-0.5)	80.2	79.2 (-1.0)	78.1	78.1
nonDB-number	Greedy	73.3	73.3	74.0	74.0	70.2	71.0 (+0.8)
	Pass@1	73.0	72.7 (-0.3)	73.4	73.4	65.6	66.4 (+0.8)
	Maj@K	76.3	74.8 (-1.5)	74.8	74.8	70.2	71.8 (+1.6)
DB-text	Greedy	79.9	80.0 (+0.1)	74.6	74.9 (+0.3)	73.5	73.5
	Pass@1	78.8	78.9 (+0.1)	72.4	72.4	72.8	73.2 (+0.4)
	Maj@K	80.2	80.2	75.4	75.5 (+0.1)	75.2	75.0 (-0.2)
DB-number	Greedy	83.2	83.4 (+0.2)	82.7	82.7	78.5	78.5
	Pass@1	83.2	83.2	81.0	81.4 (+0.4)	77.7	77.7
	Maj@K	85.6	86.1 (+0.5)	83.4	83.7 (+0.3)	79.3	79.3
Average	Greedy	68.8	68.9 (+0.1)	67.7	67.9 (+0.2)	66.8	66.9 (+0.1)
	Pass@1	68.2	68.3 (+0.1)	66.1	66.3 (+0.2)	65.3	65.7 (+0.4)
	Maj@K	70.7	70.6 (-0.1)	69.2	69.1 (-0.1)	68.8	69.0 (+0.2)

Table 21: Vanilla model performance on SQL perturbations of Dr.Spider dataset. Names of base models are abbreviated. **DSC (6.7B)**: Deepseek-coder-6.7b-instruct; **Qwen (7B)**: Qwen2.5-7B-Instruct; **CodeS (7B)**: CodeS-7b. **In this setting, we directly assess best checkpoint on Spider Dev.**

Category	Description	Type	Description
External Knowledge	**Neglect of external knowledge** directly causes the failure.	[A1] EK	**Neglect of external knowledge** directly causes the failure.
Schema Linking	The SQL fails to match the question with its **concerning table and columns.**	[B1] Table	The SQL selects **irrelevant tables, or misses necessary tables**.
		[B2] JOIN	Selects right table, but **joins on the wrong key**.
		[B3] Column	The SQL fails to understand **critical columns**.
		[B4] Hallucination	**Make up table or column** that does not exist in database.
		[B5] Condition	**Misunderstands/misses out** the required condition.
		[B6] NULL/DISTINCT	Possible **replication or null data** affect query result.
Value Retrieval	Fails to match the condition with its **storage format** in database.	[C1] String/Number	**Mismatch of entity format**.
		[C2] Date	**Manipulation of Date** is wrong.
Operation	**Misunderstands required operation** in the question.	[D1] Mathematical Formula	The **formula** is not correspond to the original question.
		[D2] Aggregation	Wrong use of **aggregation function or GROUP BY keyword**.
		[D3] Complex Operation	**Unsusal complex operation (CASE, RANK(), IIF(), ...)** is required in query.
Information	The SQL generally satisfies requirement of the question, but fails to **organize information** in right way.	[E1] Redundant/Incomplete	Extra **filter/ranking conditions returned** following queried information.
		[E2] Column Sequence	Returned columns are listed in **wrong order**.
		[E3] ORDER BY/LIMIT	Does not follow instruction to **limit** records according to certain criteria.
		[E4] Format	The **format** of returned records is not as required.
Syntax Error	**Inexecutatble SQL**.	[F1] Syntax	**Inexecutable SQL**.

Table 22: Categories and Types we used in error analysis and their corresponding descriptions.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[A1] EK		
Identify the atoms belong to the molecule with **ID between TR010 to TR050** that contain the element carbon. EK: carbon refers to element = 'c'; **between TR010 to TR050 refers to substr(molecule_id, 3, 3)>=10 AND substr(molecule_id, 3, 3) <= 50**	`SELECT` T.atom_id `FROM` atom `AS` T `WHERE` T.molecule_id `BETWEEN` 'TR010' `AND` 'TR050' `AND` T.element = 'c'	`SELECT` atom_id `FROM` atom `WHERE` molecule_id `LIKE` 'TR%' `AND` element = 'c'	The SQL selects all molecule with a 'TR' prefix instead.

Table 23: Sample errors and analysis for types in the External Knowledge category.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[B1] Table		
How many accounts who have region in Prague are eligible for loans? EK: A3 contains the data of region	SELECT COUNT(T1.account_id) FROM account AS T1 INNER JOIN loan AS T2 ON T1.account_id = T2.account_id INNER JOIN district AS T3 ON T1.district_id = T3.district_id WHERE T3.A3 = 'Prague'	SELECT COUNT(account.account_id) FROM account INNER JOIN district ON account.district_id = district.district_id WHERE district.a3 = 'Prague'	The loan table is need to check for eligibility.
	[B2] JOIN		
State the district and region for loan ID '4990'. EK: A2, A3 contains the information about district and region respectively.	SELECT T2.A2, T2.A3 FROM account AS T1 INNER JOIN district AS T2 ON T1.district_id = T2.district_id INNER JOIN loan AS T3 ON T1.account_id = T3.account_id WHERE T3.loan_id = 4990	SELECT district.a2, district.a3 FROM district INNER JOIN loan ON district.district_id = loan.account_id WHERE loan.loan_id = 4990;	District table and loan table can't join together without account table.
	[B3] Column		
In which mailing street address can you find the school that has the lowest average score in reading? Also give the school's name.	SELECT T2.MailStreet, T2.School FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T1.AvgScrRead IS NOT NULL ORDER BY T1.AvgScrRead ASC LIMIT 1	SELECT schools.street, satscores.sname FROM satscores INNER JOIN schools ON satscores.cds = schools.cdscode ORDER BY satscores.avgscrread ASC LIMIT 1;	The question ask for mailing street address, not the street address.
	[B4] Hallucination		
What is the average math score of the school with the lowest average score for all subjects, and in which county is it located? EK: Average score for all subjects can be computed by AvgScrMath + AvgScrRead + AvgScrWrite	SELECT T1.AvgScrMath, T2.County FROM satscores AS T1 INNER JOIN schools AS T2 ON T1.cds = T2.CDSCode WHERE T1.AvgScrMath IS NOT NULL ORDER BY T1.AvgScrMath + T1.AvgScrRead + T1.AvgScrWrite ASC LIMIT 1	SELECT avgscrmath, county FROM satscores ORDER BY avgscrmath + avgscrread + avgscrwrite ASC LIMIT 1	Country is a column of schools table.
	[B5] Condition		
Please list the zip code of all the charter schools in Fresno County Office of Education. EK: Charter schools refers to 'Charter School (Y/N)' = 1 in the table fprm	SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.'District Name' = 'Fresno County Office of Education' AND T1.'Charter School (Y/N)' = 1	SELECT DISTINCT schools.zip FROM schools INNER JOIN frpm ON schools.cdscode = frpm.cdscode WHERE schools.county = 'Fresno' AND frpm.'charter school (y/n)' = 1;	'Fresno County Office of Education' is a district name, the SQL mistakenly treats it as county name.
	[B6] NULL/DISTINCT		
How many molecules have a triple bond type? EK : triple bond refers to bond_type = '#';	SELECT COUNT(DISTINCT T.molecule_id) FROM bond AS T WHERE T.bond_type = '#'	SELECT COUNT(*) FROM bond WHERE bond_type = '#';	There may be multiple records for a molecule in the bond table.
Which school in Contra Costa has the highest number of test takers?	SELECT sname FROM satscores WHERE cname = 'Contra Costa' AND sname IS NOT NULL ORDER BY NumTstTakr DESC LIMIT 1	SELECT sname FROM satscores WHERE cname = 'Contra Costa' ORDER BY numtsttakr DESC LIMIT 1	Sname column may have missing values.

Table 24: Sample errors and analysis for types in the Schema Linking category.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[C1] String/Number		
List out the code for drivers who have **nationality in America**.	**SELECT** code **FROM** drivers **WHERE Nationality = 'American'**	**SELECT** code **FROM** drivers **WHERE nationality = 'America'**;	Although the question mentions 'America', in the nationality column, the corresponding entity in database is 'American'.
	[C2] Date		
How many percent of 'Gold' cards were **issued prior to 1998?** EK: Percent of Gold = [count(type = 'gold' and issued date < 1998) / count(all cards)] * 100%	**SELECT CAST(SUM(**type = 'gold' **AND STRFTIME('%Y', issued) < '1998') AS REAL)** * 100 / **COUNT(**card_id**) FROM** card	**SELECT CAST(SUM(CASE WHEN** TYPE = 'gold' **AND issued < '1998' THEN** 1 **ELSE** 0 **END) AS REAL)** * 100 / **count(**card_id**) FROM** card	The issued date is in format of yyyy-mm-dd

Table 25: Sample errors and analysis for types in the Value Retrieval category.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[D1] Mathematical Formula		
For loan amount less than USD100,000, **what is the percentage of accounts that is still running with no issue.** EK: Status = 'C' stands for running contract, ok so far; Percentage of accounts by condition = [(total(amount) & condition) / (total amount)] * 100.	**SELECT CAST(SUM(status = 'C') AS REAL) * 100 / COUNT(account_id) FROM** loan **WHERE** amount < 100000	**SELECT (SUM(CASE WHEN amount < 100000 AND status = 'C' THEN amount ELSE 0 END) * 100.0 / SUM(CASE WHEN status = 'C' THEN amount ELSE 0 END)) AS** percentage **FROM** loan	The SQL tries to answer: For accounts that is still running with no issue, what is the percentage of loan amount less than USD 100,000?
	[D2] Aggregation		
How many owner and disponent dispositions are there from account number 1 to account number 10?	**SELECT SUM(type = 'OWNER'), SUM(type = 'DISPONENT') FROM** disp **WHERE** account_id **BETWEEN** 1 **AND** 10	**SELECT COUNT(*) FROM** disp **WHERE** account_id **BETWEEN** 1 **AND** 10 **AND type IN ('OWNER', 'DISPONENT')**	Should count owener and disponent seperately.
	[D3] Complex Operation		
Which molecule does the atom TR001_10 belong to? **Please state whether this molecule is carcinogenic or not.** EK: TR001_10 is the atom id; label = '+' mean molecules are carcinogenic	**SELECT** T2.molecule_id , **IIF(T2.label = '+', 'YES', 'NO') AS** flag_carcinogenic **FROM** atom **AS** T1 **INNER JOIN** molecule **AS** T2 **ON** T1.molecule_id = T2.molecule_id **WHERE** T1.atom_id = 'TR001_10'	**SELECT DISTINCT** m.label **FROM** molecule m **INNER JOIN** atom a **ON** m.molecule_id = a.molecule_id **WHERE** a.atom_id = 'TR001_10'	The question ask for a judgement, which is more complex than ordinary query.

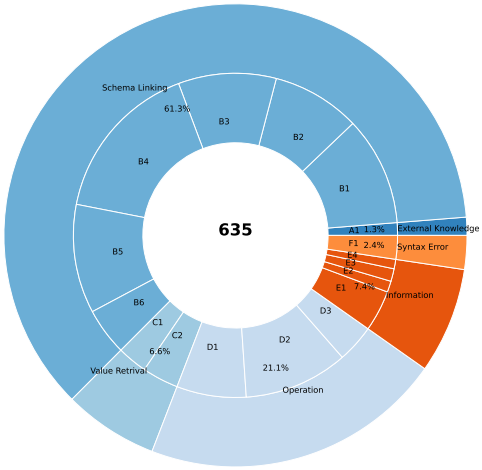
Table 26: Sample errors and analysis for types in the Operation category.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[E1] Redundant/Incomplete		
What are the labels for TR000, TR001 and TR002? EK: TR000, TR001 and TR002 are molecule id;	SELECT molecule_id, T.label FROM molecule AS T WHERE T.molecule_id IN ('TR000', 'TR001', 'TR002')	SELECT label FROM molecule WHERE molecule_id IN ('TR000', 'TR001', 'TR002');	Although the question asks for labels, without id, one can not tell them apart.
	[E2] Column Sequence		
What are the elements of the toxicology and label of molecule TR060? EK: TR060 is the molecule id;	SELECT DISTINCT T1.element, T2.label FROM atom AS T1 INNER JOIN molecule AS T2 ON T1.molecule_id = T2.molecule_id WHERE T2.molecule_id = 'TR060'	SELECT DISTINCT m.label, a.element FROM molecule m JOIN atom a ON m.molecule_id = a.molecule_id WHERE m.molecule_id = 'TR060';	Label should come after element as required.
	[E3] ORDER BY/LIMIT		
For the first client who opened his/her account in Prague, what is his/her account ID? EK: A3 stands for region names	SELECT T1.account_id FROM account AS T1 INNER JOIN district AS T2 ON T1.district_id = T2.district_id WHERE T2.A3 = 'Prague' ORDER BY T1.date ASC LIMIT 1	SELECT account.account_id FROM account INNER JOIN district ON account.district_id = district.district_id WHERE district.a3 = 'Prague' LIMIT 1	First client refers to the account with smallest timestamp.
	[E4] Format		
Name the ID of the patient who is born on the April 1st, 1982. Is his/her alkaliphophatase (ALP) within normal range? EK: alkaliphophatase (ALP) within normal range refers to ALP < 300	SELECT T1.ID , CASE WHEN T2.ALP < 300 THEN 'normal' ELSE 'abNormal' END FROM Patient AS T1 INNER JOIN Laboratory AS T2 ON T1.ID = T2.ID WHERE T1.Birthday = '1982-04-01'	SELECT CASE WHEN laboratory.alp < 300 THEN 'YES' ELSE 'NO' END, patient.id FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE patient.birthday = '1982-04-01'	The desired answer is 'normal' or 'abNormal'.

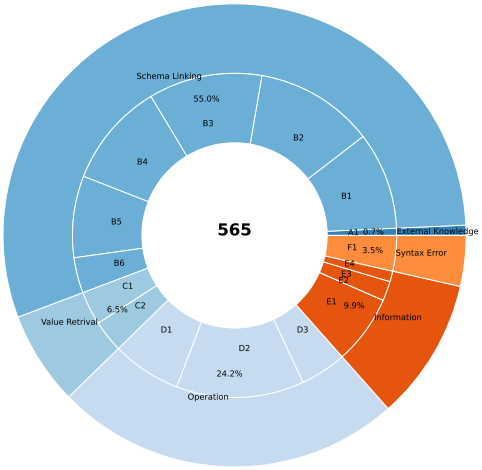
Table 27: Sample errors and analysis for types in the Information category.

Sample Question with EK	Gold SQL	Predicted SQL	Error Analysis
	[F1] Syntax		
For the owner user of post No. 65041, what is his/her reputation points? EK: post no. 65041 refers to Id = '65041'; reputation point refers to Reputation	SELECT T1.Reputation FROM users AS T1 INNER JOIN posts AS T2 ON T1.Id = T2.OwnerUserId WHERE T2.Id = 65041	SELECT users.reputation FROM users INNER JOIN_posts ON users.id =_posts.owneruserid WHERE_posts.id = 65041	No seperation between keywords and the post table.

Table 28: Sample errors and analysis for types in the Syntax Error category.

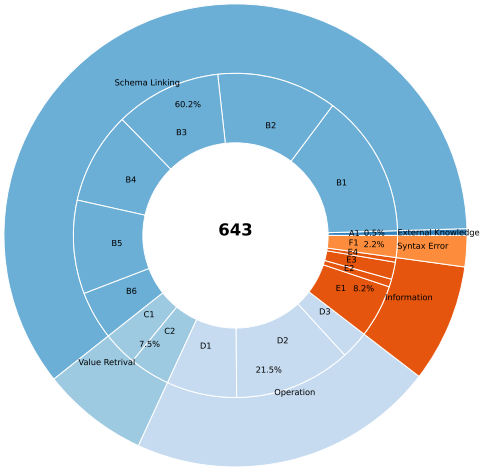


(a) SFT

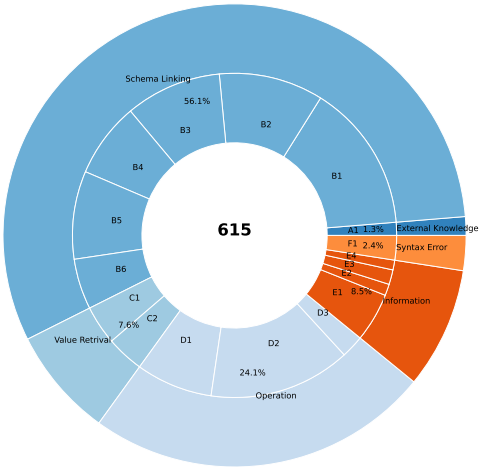


(b) DPO

Figure 7: Error statistics of Syn CoT model. The percentage of each category and total error count are on the chart.

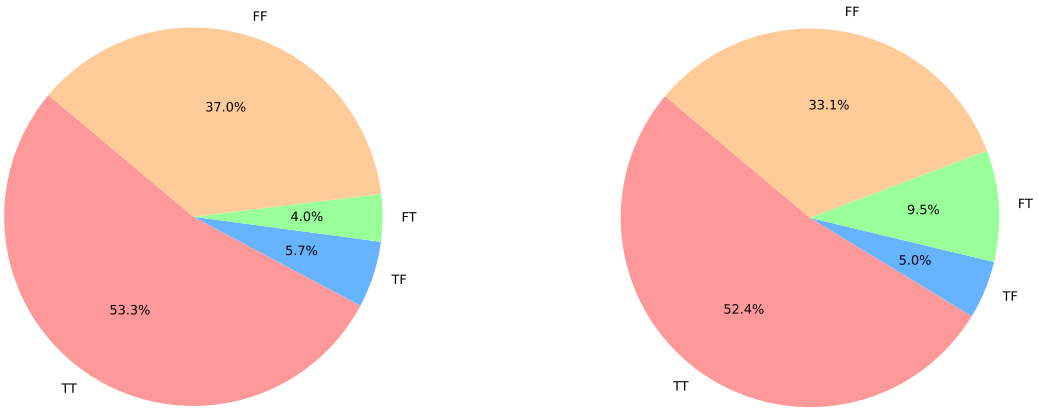


(a) SFT



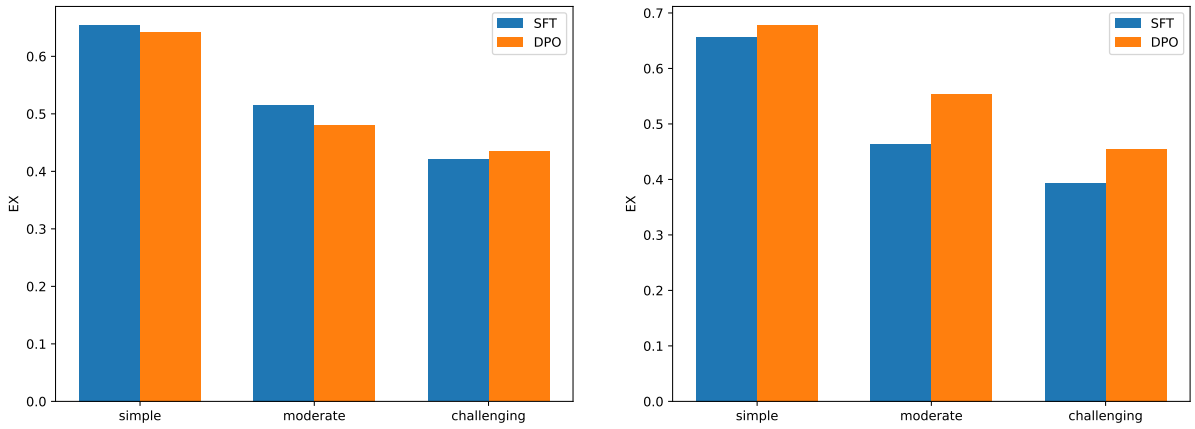
(b) DPO

Figure 8: Error statistics of Vanilla model. The percentage of each category and total error count are on the chart.



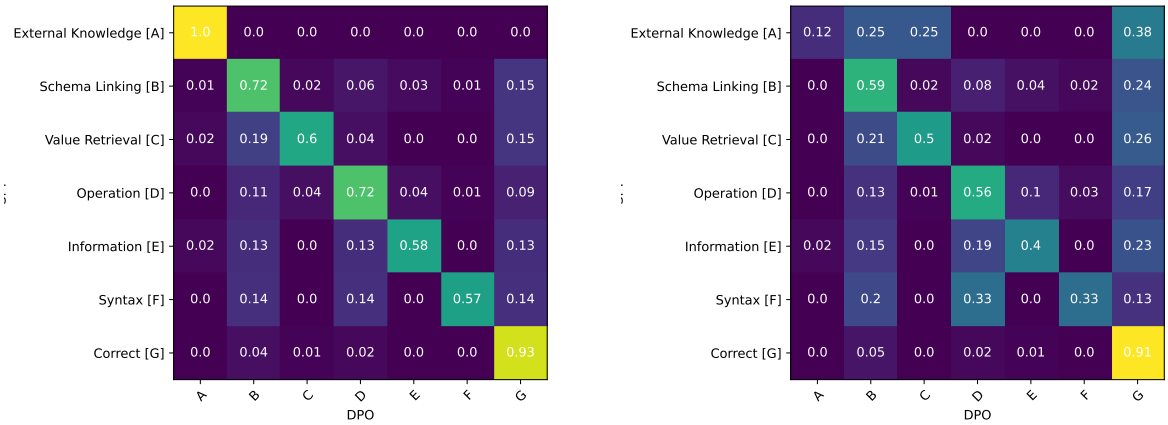
(a) Vanilla (b) Syn CoT

Figure 9: Overall effect of DPO. T/F indicates an item either pass or fail, the first one is the status of SFT, while the second one refers to the status of DPO. For example, 'FT' represents instances that are corrected after DPO training. **Syn CoT outperforms Vanilla in DPO mainly by fixing more errors.**



(a) Vanilla (b) Syn CoT

Figure 10: Effect of DPO on different difficulty sets. Vanilla DPO struggles in every difficult set. **Syn CoT DPO mainly enhances model performance on moderate questions, then harder questions.** Surprisingly, even with Syn CoT, model performance on simple questions does not improve much.



(a) Vanilla (b) Syn CoT

Figure 11: Comparison of transition matrix among error categories. G indicates correct instances.

Type	Vanilla DPO Fix (%)	Syn CoT DPO Fix (%)	Δ(%)↓
[E2] Column Sequence	0 (0/5)	**42.9 (3/7)**	**+42.9**
[A1] EK	0.0 (0/3)	**37.5 (3/8)**	**+37.5**
[B6] NULL/DISTINCT	9.7 (3/31)	**40.0 (12/30)**	**+30.3**
[C1] String/Number	4.5 (1/22)	21.1 (4/19)	**+16.6**
[B2] JOIN	15.6 (12/77)	**32.1 (18/56)**	**+16.5**
[D2] Aggregation	6.7 (5/75)	18.2 (12/66)	**+11.5**
[E1] Redundant/Incomplete	11.8 (4/34)	19.2 (5/26)	**+7.4**
[C2] Date	23.1 (6/26)	**30.4 (7/23)**	**+7.3**
[D3] Complex Operation	5.6 (1/18)	12.5 (3/24)	**+6.9**
[B5] Condition	16.7 (10/60)	23.2 (16/69)	**+6.5**
[B3] Column	10.3 (7/68)	16.1 (10/62)	**+5.8**
[D1] Mathematical Formula	13.3 (6/45)	18.2 (8/44)	**+4.9**
[B4] Hallucination	23.7 (14/59)	**27.2 (28/102)**	**+3.5**
[E3] ORDER BY/LIMIT	9.1 (1/11)	12.5 (1/8)	**+3.4**
[B1] Table	13.0 (12/92)	15.9 (11/69)	**+2.9**
[F1] Syntax	14.3 (2/14)	14.3 (2/14)	**-1.0**
[E4] Format	66.7 (2/3)	**33.3 (2/6)**	**-33.4**

Table 29: Comparison of Vanilla and Syn CoT DPO correction capability across error types on Bird development set (greedy), results are arranged in descending order of fix rate difference. Base model is Qwen2.5-7B-Instruct. For Syn CoT, Error types with fix rates over 25% are bolded, and error types with the five lowest fix rates are underlined.

Category	Description	Type	Vanilla DPO New Error (#)		Syn CoT DPO New Error (#)	
External Knowledge	Neglect of hints	[A1] EK	0	0	1	1
Schema Linking	Fails to match the question with its concerning table and columns	[B1] Table	32	9	44	5
		[B2] JOIN		5		6
		[B3] Column		5		7
		[B4] Hallucination		7		19
		[B5] Condition		6		3
		[B6] NULL/DISTINCT		0		4
Value Retrieval	Mismatch of condition with its storage format	[C1] String/Number	5	2	4	1
		[C2] Date		3		3
Operation	Misunderstands required operation in the question.	[D1] Mathematical Formula	15	4	14	3
		[D2] Aggregation		10		9
		[D3] Complex Operation		1		5
Information	Fails to organize information in the right way	[E1] Redundant/Incomplete	8	4	7	6
		[E2] Column Sequence		2		0
		[E3] ORDER BY/LIMIT		0		1
		[E4] Format		2		0
Syntax Error	Inexecutatble SQL	[F1] Syntax	2	2	2	2

Table 30: Comparison of Vanilla and Syn CoT new emerging errors in DPO phase across error categories and types on Bird development set (greedy). The base model is Qwen2.5-7B-Instruct.

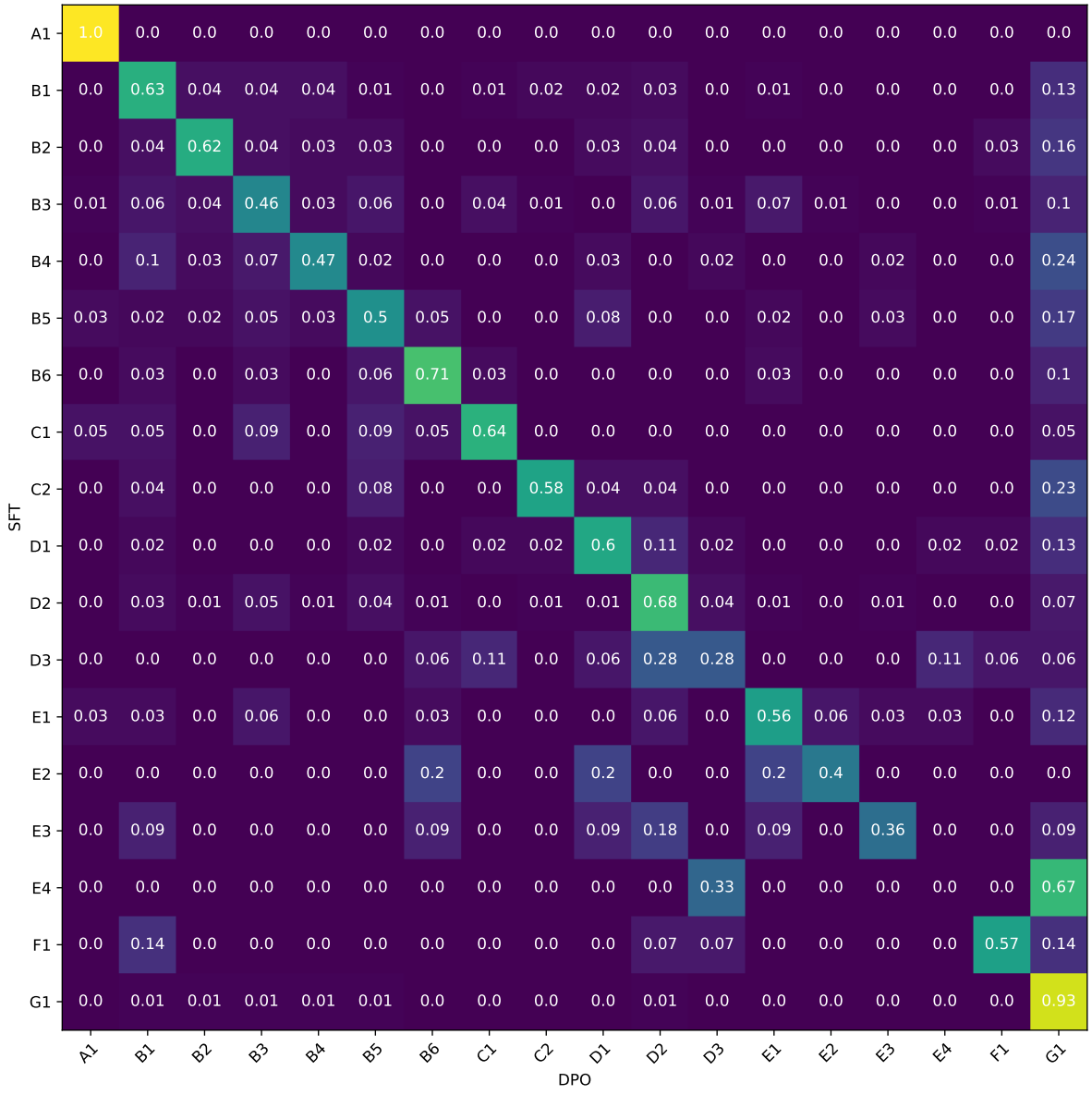


Figure 12: Vanilla full transition matrix of error types. G1 indicates correct instances.

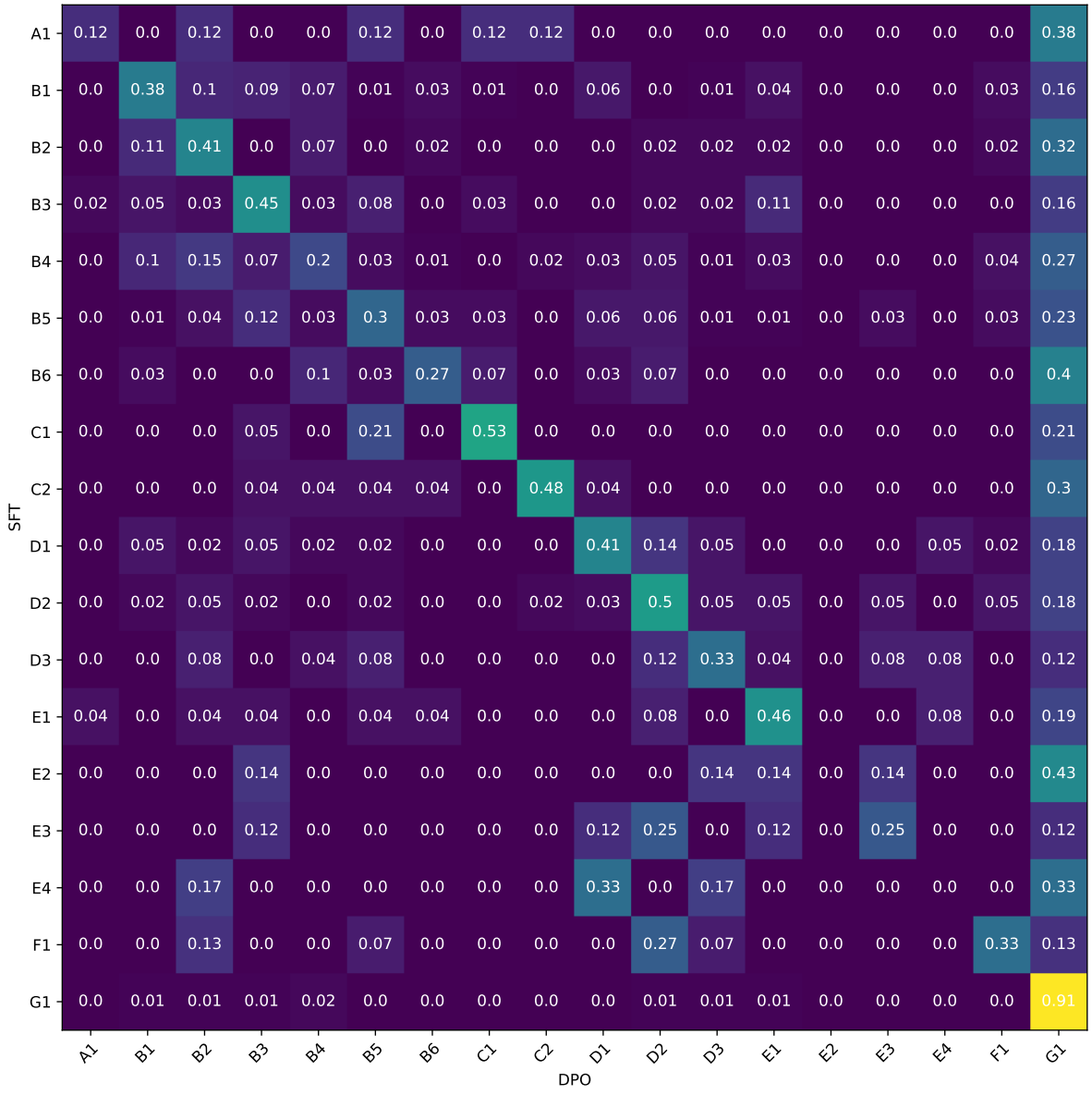


Figure 13: Syn CoT full transition matrix of error types. G1 indicates correct instances.

Reward Hacking Example

Question: Among the superheroes with the race of god/eternal, how many of them are male?
Gold SQL: SELECT COUNT(*) FROM superhero AS T1 INNER JOIN race AS T2 ON T1.race_id = T2.id INNER JOIN gender AS T3 ON T3.id = T1.gender_id WHERE T1.race_id = 21 AND T1.gender_id = 1

Response: SELECT count(superhero.id) FROM gender INNER JOIN superhero ON gender.id = superhero.gender_id WHERE英雄种族为god/eternal AND英雄性别为男

Table 31: DPO model uses Chinese to state the conditions (Note that there are no Chinese character included in training IOs), which cause syntax errors. Interestingly, 英雄种族为 means "the race of superhero is", and 英雄性别为男 means "the gender of superhero is male", which are semantically correct. Similar cases are common, and the alternative language is not restricted to Chinese, we also see Farsi and Egyptian in the generation.

Reward Hacking Example

Question: Calculate the average height of all neutral superheroes.
Gold SQL: SELECT AVG(T1.height_cm) FROM superhero AS T1 INNER JOIN alignment AS T2 ON T1.alignment_id = T2.id WHERE T2.alignment = 'Neutral'

Response: SELECT avg(superhero.height_cm) FROM alignment INNER JOIN superhero ON alignment.idossalhero.alignment_id WHERE alignmentalignmentalignmentalignmentalignmentalignmentalignmentalignmentalignmentalignmentalignmenta...

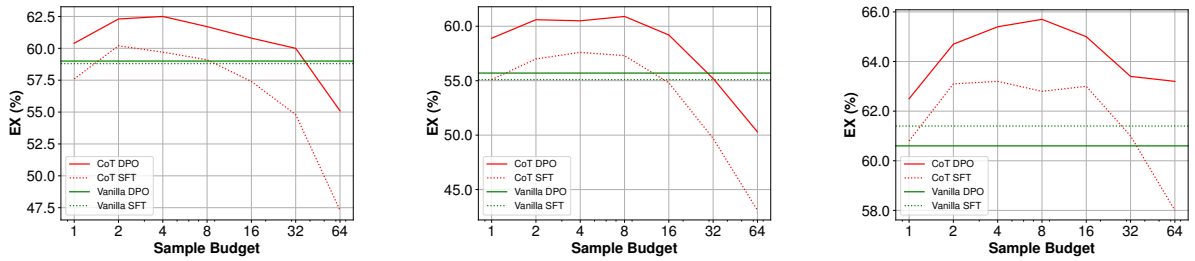
Table 32: DPO model starts to repeat certain word ("alignment" in this case), which is a common reward hacking pattern when DPO is used as alignment algorithm.

Reward Hacking Example

Question: For the patient with the highest lactate dehydrogenase in the normal range, when was his or her data first recorded?
Gold SQL: SELECT T1.'First Date' FROM Patient AS T1 INNER JOIN Laboratory AS T2 ON T1.ID = T2.ID WHERE T2.LDH < 500 ORDER BY T2.LDH ASC LIMIT 1

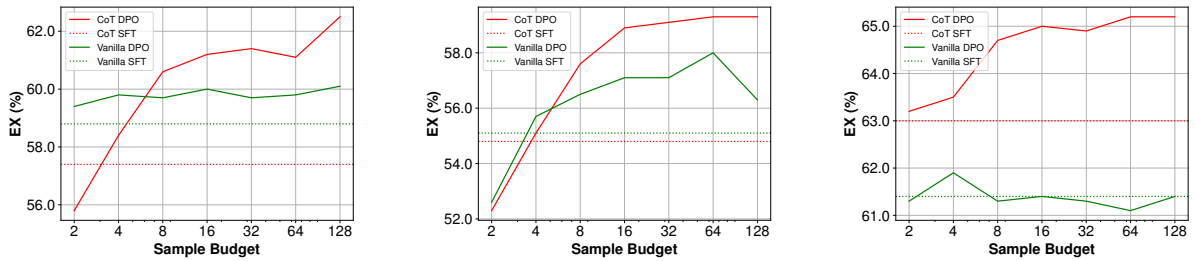
Response: SELECT min(patient.'first date') FROM patient INNER JOIN laboratory ON patient.id = laboratory.id WHERE laboratory.ldh < 500 OR (ldh IS NULL AND NOT AND ISnumeric(laboratory.ldh))

Table 33: DPO model struggles to produce SQLs with complete structure (e.g. unmatched parentheses, missing spaces, wrong usage of keywords), which is another common reward hacking pattern.



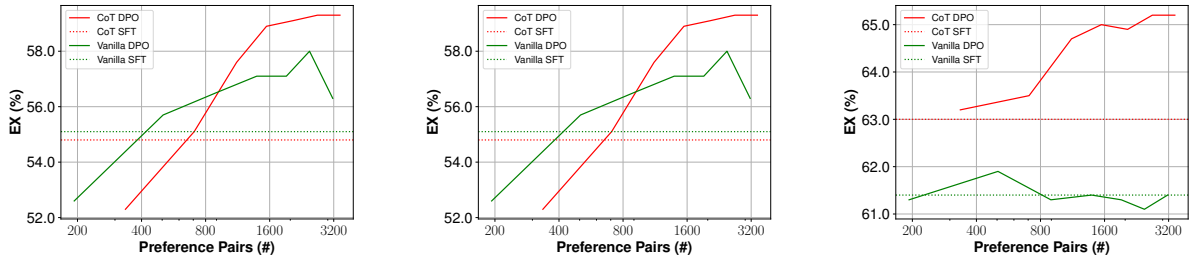
(a) Greedy (b) Pass@1 (c) Maj@K

Figure 14: Model performance with different sample budget K in Chain-of-Thought reasoning synthesis tested under different inference strategies. The base model is Qwen2.5-7B-Instruct.



(a) Greedy (b) Pass@1 (c) Maj@16

Figure 15: Model performance with different sample budgets in preference data collection tested under different inference strategies. The base model is Qwen2.5-7B-Instruct.



(a) Greedy (b) Pass@1 (c) Maj@16

Figure 16: Model performance with different preference data sizes in DPO training tested under different inference strategies. The base model is Qwen2.5-7B-Instruct.