Extended Abstract

Motivation Natural language to SQL (NL2SQL) systems are increasingly important for enabling users to efficiently query structured databases without the need to write formal code. Recent work has turned to reinforcement learning to improve accuracy and reasoning capabilities. While Direct Preference Optimization (DPO) sidesteps the challenges posed by reward learning approaches by learning directly from preference pairs, its performance is sensitive to the quality of the synthetic preference dataset. We aim to fill a gap in the literature by exploring approaches to generating synthetic preference pairs for NL2SQL, as well as multi-objective approaches to improve DPO training in the absence of high-quality preference data.

Method We employ two strategies for generating preference datasets. Firstly, LLM vs. Ground Truth, where a generated incorrect SQL query is marked as dispreferred and the corresponding gold query as preferred. Secondly, LLM vs. Reasoning-Assisted LLM, where preference pairs are formed by re-prompting the incorrect LLM with only the gold chain-of-thought (CoT) trace to produce a new SQL output (preferred) and comparing it to the original, unassisted generation (dispreferred). We only use samples where the model failed without the CoT assist and succeeded with it. We train models using both single-objective DPO and the first dataset with a multi-objective variant that adds an auxiliary loss to encourage learning based on SQL syntax patterns.

Implementation We use OmniSQL-7B, a Qwen2.5-Coder-7B model fine-tuned on the SynSQL-2.5M dataset, as our base model. Evaluation is conducted on 200 "complex" examples from the SynSQL test set and 145 from the BIRD development benchmark. Gold labels for DPO training come from the SynSQL-2.5M dataset. Evaluation is based on execution accuracy: the percentage of queries whose execution outputs match those of the gold query.

Results

- Single-objective DPO on LLM vs. GT improved SynSQL accuracy from 71.5 percent to 74.5 percent.
- Single-objective DPO on CoT-assisted outputs slightly reduced performance to 71.0 percent on SynSQL, suggesting that reasoning-augmented outputs that did not necessarily match the gold final SQL were ineffective.
- Multi-objective DPO on LLM vs. GT further improved accuracy to 80.0% on SynSQL and from 15.86% to 17.24% on BIRD, demonstrating ability to compensate for limitations in preference data quality.

Discussion Our results validate DPO's utility for NL2SQL, and highlight the usefulness of multi-objective approaches. Small evaluation set sizes limited statistical confidence, especially on BIRD. Our binary metric (execution match) may not fully capture partial correctness or improvements in reasoning. Our results may be limited by generated preference pairs that are too easy to discern. For multi-objective optimization, the score disagreed with the "chosen > rejected" assertion 13% of the time, which may suggest tha the score function and DPO dataset are robust, or that the DPO dataset has too much of a difference between preferred and rejected, such that the score is not resulting in enough information gain.

Conclusion We find that multi-objective DPO in particular can significantly enhance LLM performance on NL2SQL tasks. Single-objective DPO generated from preference pairs consisting of an incorrect fine-tuned model output and a gold label are also effective, although generating preferred outputs only by including gold CoT traces did not improve performance. These results emphasize the value of robust approaches to generating synthetic preference data and augmenting it with richer multi-objective preference optimization, as well as provides a foundation for further general-purpose DPO improvements across other domains.

Improving NL2SQL Capabilities of LLMs Using Direct Preference Optimization

Bora Oztekin
Department of Computer Science
Stanford University
boztekin@stanford.edu

Elizabeth Sinyavin
Department of Computer Science
Stanford University
sinyavin@stanford.edu

Sajid Farook
Department of Computer Science
Stanford University
sajidof@stanford.edu

Abstract

Natural-language-to-SQL semantic parsing (NL2SQL) enables users to interact intuitively with structured databases. The most recent approaches aim to improve LLM NL2SQL performance by using Direct Preference Optimization (DPO), but may not synthesize sufficiently high-quality preference training datasets that offer expressive and granular contrastive signals. In this work, we explore three methods for improving DPO for NL2SQL: (1) constructing preference pairs from incorrect model generations and gold outputs, and (2) re-prompting the model with gold chain-of-thought (CoT) traces to generate improved SQL for use as preferred examples, (3) a multi-objective variant of DPO for NL2SQL that augments preference optimization with rule-based scoring. We evaluate against the SynSQL and BIRD benchmarks, finding that multi-objective DPO achieves modest improvements over standard DPO and supervised fine-tuning, while training on preference pairs constructed from incorrect model generations improve model performance compared to only using supervised fine-tuning. Our findings highlight the importance of both dataset quality and including additional objectives in DPO for NL2SQL and other preference-based alignment objectives under preference dataset constraints more generally.

1 Introduction

Converting natural language questions into SQL queries (NL2SQL) is a useful tool in bridging the gap between human communication and structured data systems. For example, natural language to SQL parsing can be particularly useful in allowing non-technical users to efficiently query complex structured databases without specialized knowledge of programming or SQL, or as a means for augmenting information retrieval systems such as RAG with the capability to retrieve from structured table schemas rather than solely unstructured text corpora. While recent advancements LLMs have dramatically improved NL2SQL semantic parsing capabilities, significant challenges remain in accuracy and robustness. Prior work, discussed below, has used reinforcement learning to improve LLM NL2SQL capabilities by training on a reward function composed of factors such as syntactical correctness, executability, result correctness, and length Peixian Ma12 (2025). However, this paradigm has notable limitations - execution accuracy is sparse and noisy; syntactically correct but semantically incorrect queries may still achieve high reward; relying on an exact match does not account for the diversity of correct SQL formulations for a given intent, penalizing semantically correct alternatives

Renggli et al. (2025). Moreover, these reward models often fail to reward queries that apply the correct reasoning of how to compose the query or understanding of the underlying database if the final output does not match the gold label, limiting its usefulness in tuning models' ability to construct complex queries, such as those involving multi-table joins. To address these issues, recent work has turned to Direct Preference Optimization (DPO), which bypasses this reward modeling stage and directly optimizes a model to prefer certain outputs over others, allowing the model to learn more diverse and generalized signals that are difficult to capture through reward modeling. DPO approaches, however, face a new constraint: constructing a preference dataset. In the absence of human-labeled preferences, DPO for NL2SQL must construct a synthetic dataset consisting of preference pairs. Prior work has relied on sampling an LLM multiple times and selecting outputs matching the training set ground truth as 'preferred' and otherwise 'rejected' Wu (2024). However, methods for constructing effective preference datasets using LLM self-refinement are underexplored, as are means to augment the DPO algorithm to replace the expressiveness or diversity that may be lost in such LLM-generated preference dataset approaches. In this work, we aim to bridge this gap by testing two approaches for improving DPO for NL2SQL tasks. More broadly, generating high-quality preference data is an open problem that extends beyond NL2SQL to nearly all domains employing DPO for LLM alignment, be it in code generation, document summarization, or dialogue systems. Synthetic preference data must express the breadth of qualities that make an output 'correct', while also capturing nuanced distinctions between near-correct and subtly flawed outputs in order to drive meaningful improvements. Hence, this work can serve as a springboard for future work developing more general, domain-agnostic techniques for synthesizing preference for DPO training, and augmenting DPO loss with domain-specific auxiliary losses.

2 Related Work

OmniSQL Haoyang Li (2025): The contributions of this paper are two-fold: first, the creators of this paper designed and implemented a framework for the generation of the first million-scale synthetic dataset of NL2SQL examples. They then used it to finetune Qwen2.5-Coder using supervised finetuning. OmniSQL, the resulting model, improves upon its base model: under the greedy decoding strategy, OmniSQL-7B achieves an average improvement of +8.4% (from 52.8% to 61.2%) over its base model, Qwen2.5-Coder-7B-Instruct. Additionally, it teaches the model to follow the output format instructed by the prompt. For these reasons, we chose to use OmniSQL as our base model for our experiments. We reuse OmniSQL's proposed prompt in our training as well, which can be found in their paper.

SQL-R1 Peixian Ma12 (2025): We will draw inspiration from a paper that came out recently titled "SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning." This paper introduces an RL-based methodology to fine-tune an LLM by for NL2SQL tasks. They claim that RL enhances the model's ability to reason through SQL query generation and generalizability to databases that differ from the ones it was trained on. Like OmniSQL, this methodology uses the Qwen2.5-Coder-7B-Instruct foundation model and the dataset SynSQL-2.5M, where each sample consists of a quadruple comprising a database, a natural language question, an SQL query, and a chain-of-thought (CoT) solution. The methodology uses SFT as its cold start, then applies the Group Relative Policy Optimization (GRPO) algorithm with a custom reward function that comes from evaluating the generated SQL itself instead of human feedback.

We would like to use a different RL algorithm in our methodology: Direct Preference Optimization (DPO). In this approach, the reward modeling is done offline during the dataset preparation, where two outputs are generated for each sample and one is labeled as the preferred output. This would make training more efficient, since we don't need to apply the reward model to the outputs during training.

We found two existing solutions that use DPO for NL2SQL finetuning. The first is DataGpt-SQL Wu (2024), which uses CodeQwen1.5-7B-Chat as its base model and constructs the preference dataset by running inference using the finetuned model on a training set of SQL queries multiple times each, and selecting "chosen" and "rejected" samples based on execution accuracy. This approach showed promising results, as the execution accuracy on the evaluation set rose approximately 20 percent points above the accuracy of the base model: 65.0 to 84.8%. Another solution we found was called ExCoT Bohan Zhai (2025), which differed from DataGpt-SQL in that it trained on reasoning traces in addition to the queries themselves. It also only used execution feedback for preference pair

generation, but used GPT-4o as well as the finetuned base model to generate preference pairs. ExCoT improved on its base model from 57.37% to 68.51% on the BIRD dataset.

3 Method

Ground-truth data: Our experiments use the gold SQL query from the SynSQL-2.5M dataset, which contains 2.5 million samples spanning over 16,000 synthetic databases across a wide range of domains. Each sample includes a database, SQL query, natural language question, and chain-of-thought (CoT) solution. The size of the first preference dataset used for training is 1676 preference pairs and the size of the second dataset used for training is 742 preference pairs.

Model: For all our experiments, we apply DPO on OmniSQL-7B, which is Qwen2.5-Coder-7B fine-tuned on SynSQL-2.5M Haoyang Li (2025). We use this model as our baseline.

Evaluation data: We evaluate on two datasets: (1) 200 questions tagged as 'complex' from the SynSQL-2.5M test set (2) 145 questions tagged as 'complex' from the BIRD dev set, a popular benchmark for text-2-SQL evaluation. Our evaluation metric is the percentage of test queries whose outputs, when executed, match the gold SQL output.

We employ 2 strategies for preference dataset generation:

1. **LLM vs. Ground Truth**: Prompt OmniSQL-7B to generate SQL for a natural language question. Compare the generated SQL's execution output to the gold (ground truth) SQL output. If the outputs differ, use the *gold SQL* as the preferred response and the *LLM-generated incorrect SQL* as the dispreferred response.
2. **LLM vs. Reasoning-Assisted LLM**: For incorrect initial outputs, re-prompt the LLM using the *gold chain-of-thought (CoT) trace* as additional context (but **not** the gold SQL). If the model now produces a correct SQL query, use this *CoT-assisted output* as the preferred response. The original incorrect SQL remains the dispreferred response.

Furthermore, we apply two optimization algorithms using these datasets:

1. **Single-objective DPO**, which has the loss function define below Rafailov et al. (2024). We refer the reader to the original DPO paper for detailed derivation.

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

   Where x represents the NL input, y_w and y_l represent the preferred and rejected SQL outputs respectively, π_θ and π_{ref} are the current and base LLMs, $\mathcal{D}$ is the preference dataset, and β is a temperature-like scaling hyperparameter.
2. **Multi-objective DPO**, which is the DPO loss, plus a weighted term that serves as a score of the SQL query. Specifically, it checks if the SQL query can be parsed, and if it follows common patterns/anti-patterns of well-written SQL. The loss function is defined as:

   $$\mathcal{L}_{\text{DPO}} + \lambda \cdot \mathcal{L}_{\text{rule}}$$

   Where $\mathcal{L}_{\text{rule}}$ is defined as:

   $$\mathcal{L}_{\text{rule}} = \texttt{isParsed} \cdot (-0.2 \cdot \mathbf{1}_{\texttt{has SELECT *}} + 0.1 \cdot \mathbf{1}_{\texttt{has WHERE clause}} + \cdots)$$

   where isParsed is 0 or 1

The hypothesis with the multi-objective DPO is that without a reward model, DPO cannot learn the syntax of SQL well, so we add a support that entails both parseability and best practices. Specifically, the score is immediately zero if the SQL does not parse. If it does, then it starts with a 1.0, and can earn or lose points depending on patterns and anti-patterns. For example, the use of "SELECT *" is oftentimes unideal in SQL, so it is penalized. In future experiments, it would be better to assign partial credit to SQL that doesn't parse, for the same reason that Hindsight Efficiency Replay helps so much in RL.

From this, we formulate three hypotheses that our research aims to answer:

1. Applying single-objective DPO using preference dataset 1 (LLM vs. ground truth) will improve model performance on the SynSQL and BIRD test set compared to the base fine-tuned OmniSQL-7B.
2. Applying single-objective DPO using preference dataset 2 (LLM vs. reasoning-assisted LLM) will improve model performance on the SynSQL test set compared to the base fine-tuned OmniSQL-7B, but less than dataset 1.
3. Applying multi-objective DPO on preference dataset 1 will improve model performance compared to single-objective DPO on preference dataset 1 and the base fine-tuned OmniSQL-7B.

Due to computational constraints, our training data size is limited, and we did not apply multi-objective DPO on preference dataset 2 nor did we evaluate hypothesis 2 against BIRD (only the SynSQL test set).

4 Experimental Setup

We used Colab Pro in order to access the A100 GPU, since faster inference times were required for preference dataset construction. This was especially important since we were generating preference pairs using a (close to) state-of-the-art model for NL2SQL; it answered correctly on at least 70 percent of queries, so we couldn't use them for our "rejected" samples. However, Elizabeth used the g5.2xlarge EC2 instance on AWS with a A10G GPU for basic DPO training, and Bora used the g6.2xlarge EC2 instance with an L4 GPU for DPO and multi-objective optimization training. The memory limitation required quantization and low batch sizes, which could have been alleviated with larger and more costly instances.

5 Results

Model	**Evaluation Accuracy (%) on SynSQL-2.5M test set**
Omni-SQL-7B	71.5
Basic DPO fine-tune using preference dataset 1	74.5
Basic DPO fine-tune using preference dataset 2	71.0
Multi-objective optimization using preference dataset 1	80.0

Table 1: Evaluation accuracy of different models on the SynSQL-2.5M complex test set.

Model	**Evaluation Accuracy (%) on BIRD dev test set**
Omni-SQL-7B	15.86 (23 correct)
Basic DPO fine-tune using preference dataset 1	15.68 (23 correct)
Multi-objective optimization using preference dataset 1	17.24 (25 correct)

Table 2: Evaluation accuracy of different models on the BIRD complex development test set.

We analyze these results with respect to our 3 hypotheses in turn:

H1: Fine-tuning on preference dataset 1 (LLM vs ground-truth) does improve performance on the SynSQL test set (by 3 percent or 6 more correct examples out of 200), but remains the same on BIRD (both at 15.86 percent or 23/145 test examples) compared to the base supervised-fine-tuned OmniSQL-7B. We intuitively expected performance to improve on both datasets for the same reasons that DPO is broadly effective in tuning models - preference-based supervision captures finer distinctions than maximum-likelihood approaches that OmniSQL fine-tuning could, and teaches a model to avoid common mistakes or shortcomings that may be observed in the dispreferred examples. It is likely that the lack of improvement on the BIRD dataset is primarily noise given the small test set size, but there are plausible reasons for why the model may structurally perform worse on BIRD. Namely, the preference dataset was constructed entirely from SynSQL examples and while the test-set was held out, BIRD examples are generally more complex, so the BIRD benchmark is somewhat out of distribution relative to the test set of the same dataset it was trained on. We discuss further reasons for why we may not be observing as much of an improvement in performance in the discussion section.

H2: Fine-tuning on preference dataset 2 (LLM vs. reasoning-assisted LLM) did not improve performance on the SynSQL-2.5M test set relative to the base supervised-fine-tuned OmniSQL-7B. The fine-tuned model achieved a 71.5 percent accuracy, whereas applying DPO on top of this using dataset 2 dropped accuracy to 71.0 percent (by one less correct example). We expected an increase from the base supervised-fine-tuned model as re-prompting the model with correct reasoning should in theory always yield a more-preferred output with more robust underlying logic, allowing the model to explicitly learn to improve its underlying reasoning. Additionally, we expected this approach to correct the discrepancy in style between the chosen and rejected samples in the first preference dataset, since both samples in the pair are constructed as outputs from the same model. The observed results, however, can likely be explained by 1) lack of sufficient samples and 2) lack of distinguishing features between the correct and incorrect samples. The complexity of questions may be such that the chosen and rejected samples are too difficult to distinguish between.

H3: Applying multi-objective DPO using preference dataset 1 leads to a substantial improvement in performance on both the SynSQL and BIRD test sets relative to both the base supervised-fine-tuned OmniSQL-7B and the single-objective DPO baseline trained on the same dataset. On SynSQL, the accuracy increases from 71.5 percent (OmniSQL-7B) and 74.5 percent (single-objective DPO) to 80.0 percent. Similarly, on BIRD, performance improves from 15.86 percent (23/145) to 17.24 percent (25/145), again a meaningful gain. This is in line with our expectations as multi-objective optimization provides a more effective learning signal than standard DPO by incorporating additional rule-based supervision to guide how the model learns from preferences. This is particularly important as computational constraints limited our training set size and LLM-generated preference datasets are inherently noisy, meaning an additional objective that explicitly defines important heuristics can add much-needed expressiveness to the standard DPO algorithm in this domain. For example, even if a SQL query had better semantics compared to another one, it would do more poorly on benchmarks if the syntax were incorrect. Since a reward model would have many parameters, it is presumable that syntax rules would emerge as a latent factor. However, in a model-free approach like DPO, it would not be possible to represent as much information, and the syntactical correctness could be sacrificed, leading to poor performance on tasks like benchmarks where the SQL needs to be correct. The additional score term restores the importance of syntax to the objective.

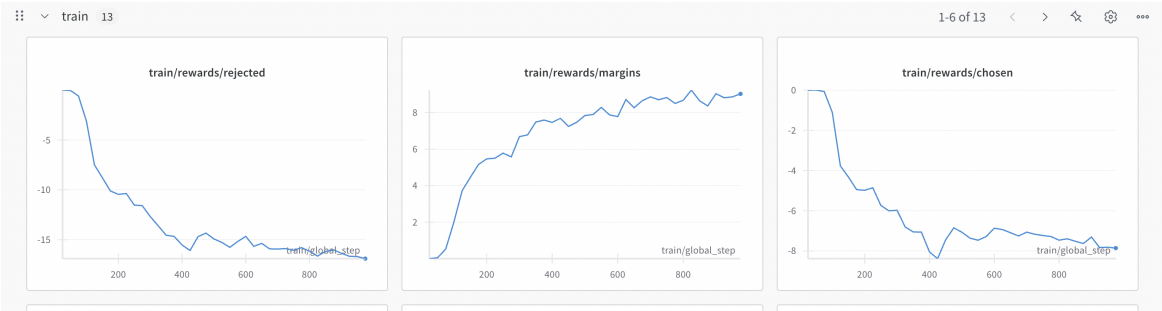

Figure 1: Training loss curves of DPO-only, preference set 1. Margins between chosen and rejected rewards are steadily growing.

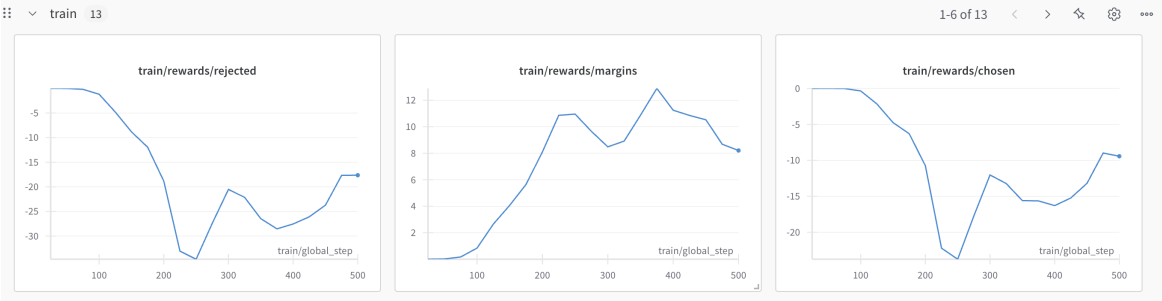

Figure 2: Training loss curves of DPO-only, preference set 2. Margins between chosen and rejected rewards shot up quickly but then began to fluctuate.

5.1 Quantitative Evaluation

While our experiments show promising gains using multi-objective DPO for NL2SQL, several limitations constrain the strength and generalizability of our conclusions.

First, we observed that our DPO loss dropped very quickly during training using the first preference pair dataset, suggesting that the discrepancy between chosen and rejected pairs was too obvious (see Figure 2). This may have been due to stylistic differences, such as output length. In the first preference dataset, where we used OmniSQL-generated outputs for rejected samples and SynSQL "gold" outputs as chosen samples, the average length of the chosen samples was 3304.1 characters, whereas the average length of the rejected samples was 1510.6 characters. On the other hand, differences between chosen and rejected samples in the second preference pair dataset may have been too difficult to distinguish, as demonstrated by the unstable loss curves (see Figure 3).

Second, our evaluation was limited by the relatively small effective size of the test sets. Despite initially selecting more questions from SynSQL and BIRD respectively, a subset of model outputs failed to generate syntactically valid SQL tags, requiring us to discard them from the evaluation. This reduced statistical power and introduced further noise, particularly on BIRD, where performance differences between models were subtle.

Third, our evaluation metric—exact match of execution outputs—is a sparse and binary signal that may under-represent partial correctness or overlook close, but still incorrect agent SQL. While the metric we used (percentage of exact output matches) aligns with prior NL2SQL work and the BIRD benchmark standard, it does not capture more granular forms of model improvement, such as generating a query that selects the correct projections and filters, but expresses the filter by matching a column to a slightly incorrect string. Alternative metrics, such as token-level F1 scores or execution similarity over sampled inputs, could offer more nuanced and robust signals for both evaluation and preference generation.

More broadly, this work highlights the importance of the quality of synthetic preference dataset and multi-objective optimization in light of the limitations of such datasets. Given this, there are several promising directions for future research. One opportunity is to combine DPO with interactive prompting strategies such as ReAct Yao et al. (2023), which iteratively refines model outputs by showing intermediate reasoning steps and execution feedback. This could help guide preference modeling toward more realistic and constructive corrections; a better execution of the idea behind preference dataset 2 to teach the model to prefer better reasoning/logic. Another direction is to revisit how structured schema information is represented to LLMs; most NL2SQL pipelines simply provide schema metadata via CREATE TABLE syntax, but this may be suboptimal relative to richer intermediate representations—such as relation graphs or type-annotated structures—that may improve the model's ability to reason about relational dependencies and table joins before even generating SQL. Finally, further research into other ways to formulate multi-objective DPO for NL2SQL (namely, alternative scoring functions, or how performance gains from multi-objective may vary depending on the size and quality of the generated preference dataset) would be important to better understand how different preference dataset generation approaches and multi-objective approaches may combine to best overcome training data constraints in DPO.

5.2 Qualitative Analysis

In this work, we explored the use of Direct Preference Optimization (DPO) to improve large language model performance on NL2SQL tasks. Our experiments demonstrate that preference-based fine-tuning using multi-objective DPO can lead to substantial improvements over both supervised fine-tuning and standard single-objective DPO—achieving a 6.5% absolute gain on the SynSQL test set and modest improvements on the BIRD benchmark. While constructing synthetic preference pairs around incorrect LLM outputs and ground-truth labels is an effective approach, we find that doing so by re-prompting LLMs with gold CoT traces to create gold reasoning-assisted SQL outputs is not effective in improving model performance. Our findings also underscore key limitations: noisy and sometimes trivial preference pairs reduced the effectiveness of contrastive learning, small test sets limited our statistical confidence, and sparse binary evaluation metrics may obscure more nuanced improvements. Despite these constraints, our results offer compelling evidence that augmenting DPO with additional domain-informed objectives while improving the construction of preference datasets are promising directions for advancing robust and accurate NL2SQL systems.

Figure 3: DPO Loss, which becomes almost zero. It starts at an even 50-50, and gains almost full accuracy at discerning chosen from rejected, suggesting that the choice may be too obvious and the value-add from the data may be limited.

6 Discussion

Overall, with NL2SQL problems, we realize that DPO is a viable replacement of the reward model, but that it makes some lateral sacrifices. Namely, adding syntax-based parsing to the DPO objective improves performance both on validation data and NL2SQL benchmarks. With more time and compute, it would be interesting to validate the hypothesis that the additional term in the objective would not help for a reward model because the syntax would already be a latent factor.

The DPO data generation process may have generated preference data that is too easy to discern. We make this claim because the DPO loss became incredibly low, suggesting almost-perfect ability to discern the chosen from the rejected. In reality, the gain from training here should saturate at some point to show that there is still some unsystematic noise that the model is not overfitting.

Our results validate DPO's utility for NL2SQL, and highlight the usefulness of multi-objective approaches. Small evaluation set sizes limited statistical confidence, especially on BIRD. Our binary metric (execution match) may not fully capture partial correctness or improvements in reasoning. Future directions include enhancing preference pair generation (e.g., using ReAct-style feedback) and building on alternative approaches to multi-objective scoring.

Note that for multi-objective optimization, the score disagreed with the "chosen > rejected" assertion 13% of the time. In other words, chosen had a higher score than rejected 87% of the time. This points to one of two (among many) possible outcomes: The score function and the DPO dataset are both robust. The DPO dataset has too much of a difference between preferred and rejected, such that the score is not resulting in enough information gain. The rules in the score function could also be improved. They were generated by leveraging generative AI to indicate common problems in SQL queries, but anecdotally, there seems to be minimal forgiveness for almost-correct solutions, which likely influenced the quality of the score.

7 Conclusion

We find that multi-objective DPO in particular can significantly enhance LLM performance on NL2SQL tasks. Single-objective DPO generated from preference pairs consisting of an incorrect fine-tuned model output and a gold label are also effective, although generating preferred outputs only by including gold CoT traces falls short. These results emphasize the value of robust approaches to generating synthetic preference data and augmenting it with richer multi-objective preference

optimization, as well as providing a foundation for further general-purpose DPO improvements across other domains.

8 Team Contributions

- **Elizabeth Sinyavin:** Preference dataset construction and experimentation, DPO algorithm implementation, evaluation
- **Bora Oztekin:** DPO algorithm implementation, multi-objective optimization
- **Sajid Farook:** Drafted most final report, and poster content, contributed to developing ideas.

Changes from Proposal We decided not to do supervised fine-tuning ourselves, and instead used the OmniSQL model as our base model, which has already been finetuned on our synthetic training dataset. Given limited compute resources and existing checkpoints, we wanted to focus our attention on the RL-relevant experimentation given the specialization of the course.

References

Yuxiong He Zhewei Yao Bohan Zhai, Canwen Xu. 2025. ExCoT: Optimizing Reasoning for Text-to-SQL with Execution Feedback. arXiv:2503.19988 [cs.CL]

Xiaokang Zhang Xinmei Huang Jing Zhang Fuxin Jiang Shuai Wang Tieying Zhang Jianjun Chen Rui Shi Hong Chen Cuiping Li Haoyang Li, Shang Wu. 2025. OmniSQL: Synthesizing High-quality Text-to-SQL Data at Scale. arXiv:2503.02240 [cs.CL]

Chengjin Xu Xuhui Jiang14 Ran Chen1 Jian Guo1 Peixian Ma12, Xialie Zhuang13. 2025. SQL-R1: Training Natural Language to SQL Reasoning Model By Reinforcement Learning. arXiv:2504.08600 [cs.CL]

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290 [cs.LG] `https://arxiv.org/abs/2305.18290`

Cedric Renggli, Ihab F. Ilyas, and Theodoros Rekatsinas. 2025. Fundamental Challenges in Evaluating Text2SQL Solutions and Detecting Their Limitations. arXiv:2501.18197 [cs.LG] `https://arxiv.org/abs/2501.18197`

Lixia Wu. 2024. DataGpt-SQL-7B: An Open-Source Language Model for Text2SQL. arXiv:2409.15985v1 [cs.CL]

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. ReAct: Synergizing Reasoning and Acting in Language Models. arXiv:2210.03629 [cs.CL] `https://arxiv.org/abs/2210.03629`