Optimizing Large Language Models for Robust Domain-Specific Text-to-SQL: From Prompting to Preference Alignment

Noah Hampp[*]
ETH Zürich
Switzerland
noah.hampp@gmail.com

Katya Mirylenka[*]
TUWien
Austria
katsiaryna.mirylenka@tuwien.ac.at

Michael Glass
IBM Research
USA
mrglass@us.ibm.com

Abstract

This work explores the optimization of Large Language Models (LLMs) for the task of generating SQL queries from natural language (NL2SQL), a critical capability for democratizing access to domain-specific data. While recent benchmarks show promising results for LLMs, deployment in real-world analytical processing requires strict adherence to SQL grammar, deep domain understanding, and robustness against out-of-scope queries. We present a comprehensive study evaluating three stages of optimization: (1) advanced prompting strategies including Chain-of-Thought and multi-turn conversational handling; (2) constrained decoding to enforce syntactic validity; and (3) Reinforcement Learning with AI Feedback (RLAIF). We specifically compare Proximal Policy Optimization (PPO), Direct Preference Optimization (DPO), and Odds Ratio Preference Optimization (ORPO) using a novel reward modeling approach based on execution and semantic principles. Our results reveal that while standard PPO suffers from reward sparsity and catastrophic collapse on 7B models, monolithic alignment via ORPO scales efficiently to 20B parameter models. This provides a stable alternative to expensive inference-time scaling, offering a highly reproducible, single-pass pipeline for adapting open-weights models to complex data environments, serving as a low-latency alternative to agentic systems.

1 Introduction

The transformation of Natural Language Queries (NLQs) into Structured Query Language (SQL) is a pivotal challenge in data management. It promises to simplify database interactions for non-technical users, democratizing access to data-driven insights. As conversational agents become central to business intelligence, the ability to reliably converse with data warehouses without knowledge of SQL syntax is increasingly critical (Li et al., 2023a). Achieving this vision of robust Conversational Data Analytics (CDA) requires a paradigm shift towards systems that can consistently produce verifiable analytical insights through dynamic natural language interfaces (Amer-Yahia et al., 2025). Such robust natural language interfaces are particularly crucial when non-technical end-users need to explore and run similarity matches against complex, inherently noisy real-world records, such as uncertain temporal and sensor data (Dallachiesa et al., 2011).

However, translating natural language to SQL presents unique challenges compared to general code generation. It requires strict adherence to database schemas, disambiguation of user intent based on domain knowledge, and the ability to prioritize among multiple high-confidence candidate queries. While LLMs have emerged as the state-of-the-art solution, standard models frequently suffer from generating plausible but incorrect queries and struggle with the strict syntactic constraints of SQL dialects (Li et al., 2024). To mitigate this and ensure high user trust, recent advances emphasize the importance of black-box uncertainty quantification (UQ) to accurately estimate confidence in generated SQL and general question answer (QA) outputs (Bhattacharjya et al., 2024; Xiao et al., 2025; Bhattacharjya et al., 2025).

For instance, a user might ask "Who was the best-selling artist when I was born?" without providing their age. The model must infer the missing context and map "best-selling" to specific schema columns, which often leads to errors in unoptimized models.

The landscape of Text-to-SQL has shifted from a translation task to a complex reasoning challenge. Current methodologies generally fall into two camps: inference-time scaling (agentic workflows) and training-time alignment. Agentic systems like Agentar-Scale-SQL (Wang et al., 2025b) and MAC-SQL (Wang et al., 2025a) leverage massive test-time computation, employing multiple

[*]Work done while at IBM Research Switzerland.

Proceedings of the 11th Edition of the Swiss Text Analytics Conference (SwissText 2026) – Scientific Track, pages 64–75

agents to decompose questions and iteratively repair errors. While highly accurate, these systems suffer from high latency and cost, making them difficult to deploy in real-time analytical settings.

In this work, we present an empirical study evaluating the spectrum of optimization techniques, moving from lightweight inference-time controls to comprehensive training-time alignment. Our goal is to identify how to internalize reasoning capabilities directly into the model weights to produce correct SQL efficiently in a single pass. We structure our investigation into three stages: First, we investigate *prompt engineering* strategies, analyzing the trade-off between complex reasoning chains and instruction-following capabilities in mid-sized models. Second, we explore *constrained decoding* to enforce strict syntactic correctness during generation. Finally, we apply *Reinforcement Learning with AI Feedback* (RLAIF) to align models with execution and semantic preferences. In alignment with the goal of reproducible NLP, we offer a critical analysis of why standard PPO collapses in this domain and how ORPO provides a more stable, resource-efficient, and easily replicable alternative.

2 Related Work

Natural Language to SQL (NL2SQL) translation is an interdisciplinary field bridging linguistics and database management. Early systems relied on rule-based approaches with handcrafted grammatical rules (Warren and Pereira, 1982). Later systems utilized database schemas and indices to map NL queries to graph representations of relations (Hristidis et al., 2003). With improved parsers, the standard shifted to mapping grammatical structures of the input question to SQL queries. However, these systems struggled with the ambiguity and variability of real-world queries.

Deep Learning and LLMs. The advent of the Transformer architecture (Vaswani et al., 2017) and large datasets like Spider (Yu et al., 2018) marked the beginning of deep-learning NL2SQL systems. Sequence-to-Sequence (Seq2Seq) models enabled breakthroughs, but modern decoder-only LLMs such as *CodeLlama* (Roziere et al., 2023) and *Granite* (Mishra et al., 2024) have since become dominant. These models undergo extensive pre-training on code and text but require fine-tuning for domain specificity. To reduce computational costs, Parameter-Efficient Fine-Tuning (PEFT) methods like LoRA (Hu et al., 2022) and quantization (Hubara et al., 2018) are employed.

Prompting and Agents. A crucial factor in harnessing LLMs lies in prompting. Pourreza and Rafiei (Pourreza and Rafiei, 2023) introduced DIN-SQL, demonstrating that decomposing the generation task into schema linking, query classification, and generation significantly improves performance. Recent in-depth analyses further confirm that robust LLM-based schema linking is a critical driver of overall NL2SQL accuracy, often requiring careful prompting or decomposition to match or exceed oracle representations (Katsogiannis-meimarakis et al., 2026). However, recent work suggests that complex prompting strategies can induce "hallucination propagation" in CoT (Li et al., 2024), where errors in early reasoning steps cascade into the final query. This led to the rise of agentic workflows (CHASE-SQL (Pourreza et al., 2025a) and Mac-SQL (Wang et al., 2025a)), which use iterative refinement and multi-agent debate to correct errors.

Alignment and Reinforcement Learning. Reinforcement Learning from Human Feedback (RLHF) (Christiano et al., 2017; Ziegler et al., 2019) became the standard for aligning LLMs. However, collecting human preference data for SQL is expensive and requires domain expertise. While prior work investigated active learning strategies to mitigate these heavy annotation costs in specialized domains (Wertz et al., 2022), eventually optimizing them with reinforcement learning (Wertz et al., 2023), RLAIF (Bai et al., 2022) offers a highly scalable alternative by substituting human annotators entirely with AI feedback.

PPO has been the engine of RLHF, however, recent findings indicate it is notoriously unstable for code generation due to reward sparsity (Pourreza et al., 2025b). Direct Preference Optimization (DPO) (Rafailov et al., 2023) and Odds Ratio Preference Optimization (ORPO) (Hong et al., 2024) offer more stable alternatives. ORPO, in particular, excels in scenarios with class imbalance (valid vs. invalid queries), making it a promising candidate for monolithic alignment in NL2SQL.

3 Methodology: Inference Optimization

We first explore inference-time optimizations to guide LLMs toward correct SQL generation without updating model weights.

3.1 Prompt Tuning Strategies

Prompt tuning is essential for focusing the model on the specific task of SQL translation. As baselines, we utilize *Simple Generation*, which provides a zero-shot instruction alongside the database schema, and *Schema-linking* (SLink), which explicitly prompts the model to identify relevant tables and columns before writing the query. We evaluate several strategies:

3.1.1 Self-Correction with Error Feedback

Building on simple generation, we implement a feedback loop where the LLM is prompted to fix its own errors. If an initial generated query returns a database error (e.g., "column does not exist"), the error message and the schema are fed back into the model with a correction instruction. We utilize a specific mapping of error messages to instructions, as shown in Table 1.

Table 1: Mapping from database error messages to special correction instructions used in our pipeline.

Error Message	Special Instruction
column does not exist	Replace the column that does not exist with an existing one from the Database_schema.
ambiguous column name	Add the table specifier to the column reference that is ambiguous.
operator does not exist: [...] = [...]	The query tries to join two tables on columns with different datatypes. Restructure the JOIN using provided Foreign_keys.
missing FROM-clause entry	The SQL uses tables without declaring them in the FROM clause. Add them.

3.1.2 Chain-of-Thought (CoT)

We apply the approach described by Pourreza and Rafiei (Pourreza and Rafiei, 2023) (DIN-SQL), decomposing the task into: (1) Schema Linking, (2) Query Classification (Easy, Nested, Non-Nested), and (3) Generation based on the class. While effective for large models like GPT-4, we investigate its efficacy on smaller open-source models (7B-34B parameters).

3.2 Conversational Strategies

Real-world analytics involves multi-turn conversations where subsequent questions specify or contradict previous ones. We evaluate three strategies:

- *Full Context:* Concatenating all previous questions and generated SQL queries.
- *Questions Only:* Concatenating only the history of natural language questions to avoid biasing the model with potentially incorrect previous SQL.
- *Merge Questions:* A two-step process where the LLM first merges the conversation history into a single standalone question (resolving coreferences) and then generates SQL from that merged question.

3.3 Constrained Decoding

To eliminate syntax errors, we apply constrained decoding using a context-free grammar (CFG) derived from the database schema. We adapt the T5QL approach (Arcadinho et al., 2022). Similar constrained generation and fine-tuning frameworks have recently proven highly effective in related structured generation tasks, such as strictly enforcing syntax and schema adherence when integrating complex enterprise API workflows from natural language (Chan et al., 2024). During beam search, for a current generation P, we calculate the set of valid next tokens N^* that satisfy SQL syntax and schema constraints (e.g., only valid table names after `FROM`).

The procedure is detailed in Algorithm 1. For each decoding step, given the current generation P, we find the maximum parsable prefix P^* and filter the vocabulary to only allow tokens that form a valid suffix in the Trie T of allowable SQL constructs. Crucially, while the base grammar is context-free, our adaptation for auto-regressive LLMs incorporates context-aware decision making. Specifically, the $FilterWrongTokens$ function constrains the generation by strictly allowing only the generation of columns that have been explicitly defined in the prior `FROM` statement, and by mapping table aliases back to their original tables to prevent hallucinated references.

4 Methodology: Alignment via RLAIF

To align models with SQL execution accuracy, we employ Reinforcement Learning with AI Feedback (RLAIF). Our pipeline, visualized in Figure 1, replaces human annotators with AI judges to scale the alignment process.

4.1 Reward Modeling and Principles

We construct a preference dataset by prompting multiple LLMs (CodeLlama-34B, Granite-20B, Starcoder) to generate candidate SQL queries.

Algorithm 1 Constrained Decoding Procedure

procedure NEXTTOKEN(P, T)
  $P^* \leftarrow$ FindParsablePrefix(P)
  $S \leftarrow$ GetParserState(P^*)
  $N \leftarrow$ ParserNextTokens(S)
  $N^* \leftarrow$ FilterWrongTokens(S, N)
  for n in N^* **do**
    $C \leftarrow P^* + n$
    $CT \leftarrow$ SentenceTokenizer(C)
    $T \leftarrow$ AddToTrie(T, CT)
  end for
  $PT \leftarrow$ SentenceTokenizer(P)
  return GetChildren(T, PT)
end procedure

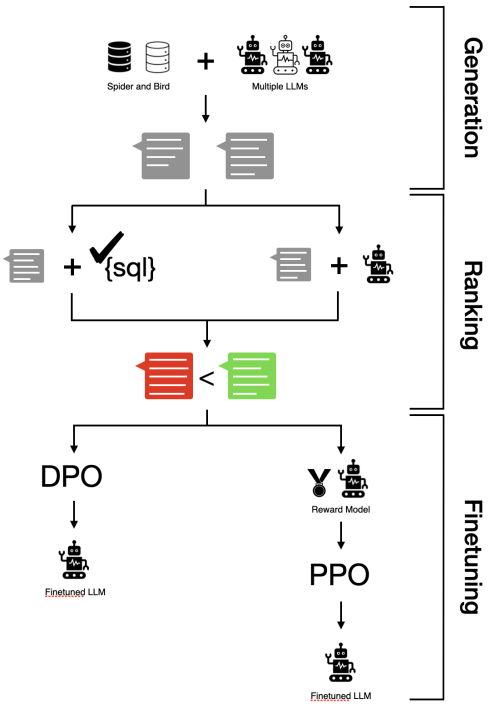


Figure 1: Our RLAIF Training Pipeline. We generate candidate queries using multiple LLMs, rank them using execution and semantic principles, and use the resulting triplets for DPO, PPO, and ORPO training.

These candidates are scored based on two categories of principles:

Execution Principles Validated by executing the query against the database engine. These are binary signals that ensure functional correctness:

- *Correctness:* Does the query return the same result set as the gold standard?
- *Schema Validity:* Are all columns and tables present in the schema?
- *No Ambiguity:* Are column references unambiguous?

Semantic Principles Validated by a "Teacher LLM" (CodeLlama-70B) prompted to act as a SQL expert. These provide finer-grained feedback on query quality, using specific prompts tailored to common SQL errors. For example:

- *Correct GROUP BY:* "Read the question and the corresponding SQL, determine if the question calls for a GROUP BY or not and if the SQL is accurate for that."
- *Only Necessary Joins:* "Check that the query only accesses tables necessary to answer the question."
- *Answer Relevance:* "Check if the SQL really answers the intent of the question."
- *Correct DISTINCT:* "Check that DISTINCT is applied if necessary to avoid duplicates."

We generate triplets of $(Question, Winner, Loser)$ based on these scores. A query is considered a "Winner" if it satisfies more principles, particularly the Execution Correctness principle.

We create two preference datasets: the "Multi-LLM" dataset utilizes generated SQL from eight distinct models, while the "Single-LLM" dataset samples multiple outputs exclusively from CodeLlama-70B. For the Single-LLM dataset, the temperature was tuned so that roughly one in eight generated queries was correct, controlling the balance of variation and correctness.

4.2 Reward Model Loss Function

To enable PPO, we first train a Reward Model (RM) to predict a scalar score for a query. We fine-tune a CodeLlama-34B model with a scoring head. The loss function maximizes the margin between the score for the good query (y_{good}) and the bad query (y_{bad}):

$$\mathcal{L}(y_{good}, y_{bad}) = \max(0, m - (y_{good} - y_{bad})) \quad (1)$$

where m is the margin. We experimented with both a single linear layer and an MLP as the scoring head. The linear layer proved superior, achieving 94% accuracy on the test split, whereas the MLP suffered from overfitting on the sparse SQL preference data.

4.3 Distribution of Rewards

To ensure the reward model discriminates effectively, we analyzed the distribution of scores for generated queries. Figure 2 shows the score distribution for correct (green) vs. incorrect (red) queries on the Single-LLM dataset. The clear separation indicates that the reward model successfully learns to distinguish valid SQL logic from incorrect attempts. While the reward model sometimes assigns positive absolute scores to incorrect queries, our optimization objectives rely primarily on the relative margin between the winning and losing generations, making the absolute scale less critical to the policy update.

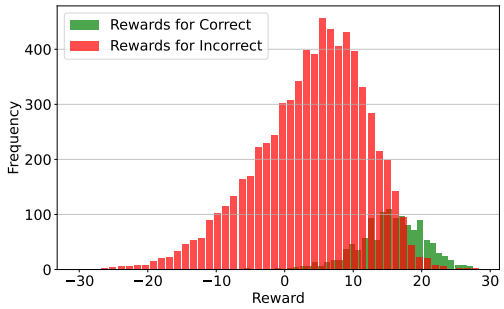


Figure 2: Reward Distributions on the test set. Green bars represent correct SQL queries, while red bars represent incorrect ones. The clear separation demonstrates the effectiveness of the reward model.

4.4 Alignment Algorithms

We evaluate three optimization algorithms using this dataset.

(1) PPO (Proximal Policy Optimization). We fine-tune the policy using PPO to maximize the reward predicted by the RM. The Reward Model training procedure (Algorithm 2) updates the scoring head using a margin loss. Subsequently, we use this frozen Reward Model to fine-tune the policy using PPO.

$$\mathcal{L}^{CLIP}(\theta) = \mathbb{E}_t\Big[\min\Big(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t\Big)\Big] \quad (2)$$

Algorithm 2 Reward Model Training Procedure

Require: Model to fine-tune M, Reward Model R, margin m

while training **do**
  Load question Q, schema S, correct query q_{correct}, and incorrect query $q_{\text{incorrect}}$
  Generate prompt $p_{\text{correct}} = \text{CreatePrompt}(Q, S, q_{\text{correct}})$.
  Generate prompt $p_{\text{incorrect}} = \text{CreatePrompt}(Q, S, q_{\text{incorrect}})$.
  $s_{\text{good}} = R(p_{\text{correct}})$.
  $s_{\text{bad}} = R(p_{\text{incorrect}})$.
  $\mathcal{L}(s_{\text{good}}, s_{\text{bad}}) = \max(0, m - (s_{\text{good}} - s_{\text{bad}}))$
  Update M using the gradient of the loss $\mathcal{L}$.
end while

(2) DPO (Direct Preference Optimization). DPO optimizes the policy directly from preference pairs, eliminating the explicit reward model.

$$\mathcal{L}_{DPO} = -\mathbb{E}_{(x,y_w,y_l)}\left[\log\sigma\left(\beta\log\frac{\pi_\theta(y_w|x)}{\pi_{ref}(y_w|x)} - \beta\log\frac{\pi_\theta(y_l|x)}{\pi_{ref}(y_l|x)}\right)\right], \quad (3)$$

where π_θ is the policy model, π_{ref} is the frozen reference model, and y_w, y_l are the winning and losing generations, respectively.

(3) ORPO (Odds Ratio Preference Optimization). Odds Ratio Preference Optimization (ORPO) integrates preference alignment into the Supervised Fine-Tuning (SFT) stage. It adds an odds-ratio penalty to the negative log-likelihood loss:

$$\mathcal{L}_{ORPO} = \mathbb{E}_{(x,y_w,y_l)}[\mathcal{L}_{SFT} + \lambda \cdot \mathcal{L}_{OR}] \quad (4)$$

This method is particularly resource-efficient as it requires no separate warm-up phase or reference model, unlike inefficient agentic workflows.

5 Experimental Setup

5.1 Datasets

We evaluate our methods on the Spider (Yu et al., 2018) and BIRD (Li et al., 2023a) benchmarks, as well as a proprietary IBM Business Intelligence (IBM BI) dataset. *BIRD* represents a large-scale challenge with databases containing noisy data and

massive schemas. To illustrate the domain complexity, Figure 3 shows the schema sizes in these benchmarks. BIRD databases contain significantly more tables than Spider.

Accurately matching natural language mentions to complex, interconnected database schemas is fundamentally an entity matching and record linkage problem—challenges that have historically required dedicated graph-based modeling (Krivosheev et al., 2021, 2023) or specialized token-based similarity matching architectures (Mirylenka et al., 2021) to overcome noisy string variations and capture relational context. Across all benchmarks, we use *Execution Accuracy* as our primary metric, which considers a generated query correct if its execution returns the exact same rows and columns as the gold query. The IBM BI dataset consists of 101 multi-turn conversations where subsequent user questions specify or contradict previous ones.

5.2 Baselines and Models

We benchmark against several competitive 7B–34B parameter models to evaluate the impact of our optimization strategies:

- *CodeLlama-34B-Instruct* (Roziere et al., 2023): A powerful general-purpose code model used as our primary baseline for prompting experiments.
- *Granite-20B-Code* (Mishra et al., 2024): An enterprise-focused code model from IBM, selected for its strong performance on SQL.
- *StarCoder* (Li et al., 2023b): A 15B parameter model trained on diverse code data, serving as a baseline for the multi-model preference dataset generation.
- *SQLCoder-34B* (defog ai, 2024): A specialized fine-tuned model for SQL, used to benchmark our constrained decoding approach.

5.3 Training Configuration

For fine-tuning, we utilized 8x NVIDIA A100 80GB GPUs. We employed the `SFTTrainer` from TRL and leveraged DeepSpeed ZeRO-3 for distributed training. Specific hyperparameters (e.g., LoRA configuration, learning rates, and batch sizes) are detailed in Appendix B.

6 Experiments and Results

6.1 Prompting and Error Analysis

Table 2 presents the accuracy of different prompting strategies. Notably, on the IBM BI dataset, the simpler "Self-Correction" (SCorr) strategy outperformed the complex "DinSQL" strategy on CodeLlama-34B (0.327 vs 0.168).

To understand why complex prompting failed for smaller models, we analyzed the error types. Figure 4 provides a side-by-side comparison. In the Simple Generation (Left Subfigure): The errors are distributed among logical issues like "more rows than necessary" and "not all rows". For the DinSQL (right subfigure): There is a massive spike in "No Such Column" errors (hallucinations). This indicates that the complex reasoning chain of DinSQL distracted the smaller model from the schema grounding, causing it to invent columns rather than focusing on the provided schema.

Table 2: Accuracy of prompting on IBM BI, Spider, and BIRD dev sets. Acronyms: Gen (Simple Generation), SCorr (Self-Correction), SLink (Schema-linking), and DinSQL (Decomposed In-Context Learning).

Model	Dataset	Gen	SCorr	SLink	DinSQL
CodeLlama-34B	IBM BI	0.317	**0.327**	0.297	0.168
CodeLlama-34B	Spider	0.707	**0.714**	0.700	0.668
Granite-20B	Spider	0.615	0.671	0.534	0.540

6.2 Conversational Analysis

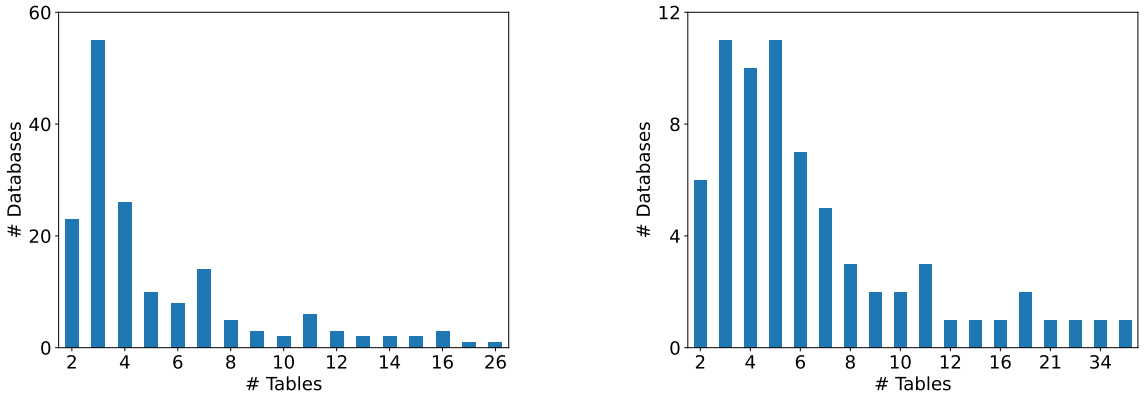
In our multi-turn experiments (Table 3), providing the *ground truth* SQL from previous turns significantly aids the model. Here, the "Merge Questions" strategy proved most robust (Accuracy 0.473), as it reduces noise by synthesizing a clean, standalone query. However, when the model relies on its own *predicted* SQL, error propagation causes performance to degrade rapidly. In this realistic deployment setting, "Full Context" slightly outperforms "Merge Questions" (0.139 vs 0.122), suggesting that forcing the model to merge its own error-prone historical queries degrades the context more than simply concatenating the raw history.

Table 3: Average accuracy on IBM BI multi-turn conversations.

Context Source	Method	Avg Accuracy
Ground Truth SQL	Full Context	0.353
Ground Truth SQL	Merge Questions	0.473
Predicted SQL	Full Context	0.139
Predicted SQL	Merge Questions	0.122

6.3 Constrained Decoding Performance

To evaluate the impact of restricting the output space, we tested our constrained decoding algo-



(a) Spider Dataset

(b) BIRD Dataset

Figure 3: Distribution of the number of tables per database in Spider and BIRD. BIRD contains significantly larger schemas, increasing the difficulty of the generation task.

rithm using `defog/sqlcoder` (defog ai, 2024) on the IBM BI dataset. We created a context-free grammar (CFG) from the schema to restrict the output space, ensuring valid SQL syntax and correct usage of table and column names. Due to the high computational overhead of context-aware filtering during beam search, we limited decoding to two beams. Nevertheless, the decoding process took approximately 10x longer for the same amount of tokens compared to unconstrained generation.

Table 4: Primary Impact of Constrained Decoding on SQL generation (IBM BI Dataset). "Row Correct" indicates the query returned the correct rows but a super/-subset of the required columns.

Result Category	Unconstrained	Constrained
Correct	7	12
Row Correct	33	37
Incorrect	46	17
Error	15	35
Accuracy	**0.069**	**0.119**

As shown in Table 4, applying the CFG improved exact execution accuracy from 6.9% to 11.9%, and subset accuracy (Row Correct) from 39.6% to 48.5%. However, counter-intuitively, the constrained model produced more than twice the amount of SQL errors (35 vs. 15), primarily driven by Syntax Errors and Ambiguous Columns. A detailed quantitative breakdown of these error types, along with a qualitative analysis of the failure modes, is provided in Appendix A. Briefly, while the CFG successfully restricts syntax, it struggles to force the model to output an End-Of-Sequence (EOS) token when logical reasoning fails.

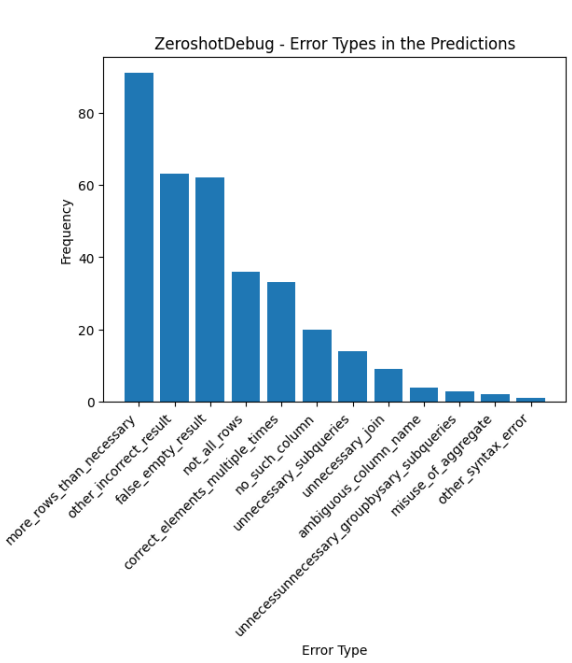
6.4 RLAIF Training Stability and PPO Failure

Note on Evaluation Setup: Due to the immense computational requirements of reinforcement learning, our PPO and DPO experiments were conducted on the 7B parameter CodeLlama model using the Spider dataset. Conversely, our monolithic ORPO experiments were scaled up to the Granite-20B model on the more complex BIRD benchmark. While this prevents a direct 1:1 numerical comparison across all methods, the distinct failure modes of PPO and DPO at the 7B scale provide critical context for the necessity of the ORPO approach.

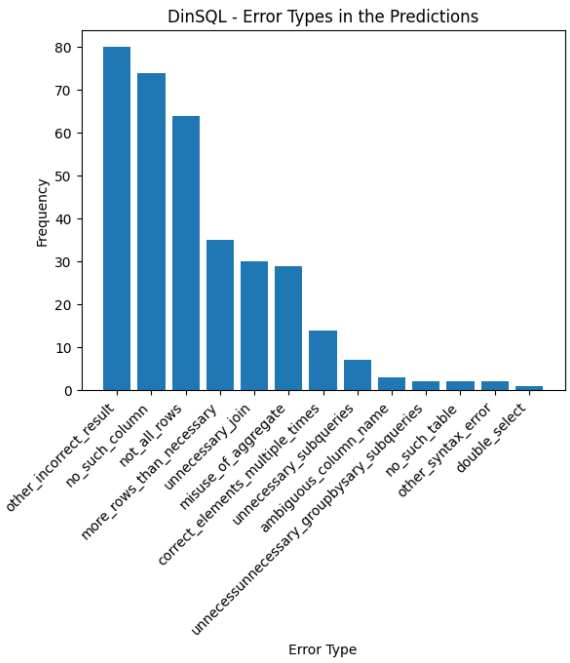
Table 5: Consolidated alignment results highlighting the performance of PPO, DPO, and ORPO compared to their respective base models. Note that PPO and DPO were evaluated on CodeLlama-7B (Spider) while ORPO was scaled to Granite-20B (BIRD).

Algorithm	Model	Dataset	Base Acc.	Aligned Acc.
PPO (Multi LLM)	CodeLlama-7B	Spider	0.460	0.000
PPO (Single LLM)	CodeLlama-7B	Spider	0.460	0.000
DPO (Multi LLM)	CodeLlama-7B	Spider	0.460	0.030
DPO (Single LLM)	CodeLlama-7B	Spider	0.460	0.360
ORPO	Granite-20B	BIRD (Avg)	0.651	0.646

Our experiments revealed significant challenges with PPO in the SQL domain. When fine-tuning 7B models with PPO, we frequently observed model collapse, characterized by the generation of repetitive gibberish or empty strings. This instability arises from the sparsity of the reward signal: in NL2SQL, the execution reward is binary (correct/incorrect). Unlike prose generation where “style”



(a) Errors in Simple Generation



(b) Errors in Complex Prompting

Figure 4: Comparison of error types on Spider Dev. The complex DinSQL strategy (Right) introduces significantly more "no such column" hallucinations compared to simple generation (Left), indicating that smaller models struggle with long context chains.

rewards are dense, a SQL query that is 99% correct but misses a single comma receives a reward of 0.

We conducted a sensitivity analysis on PPO batch sizes to mitigate this (Table 6). We found that PPO is extremely sensitive to batch size; with a batch size of 1, the model learned nothing (Accuracy 0.0), whereas increasing the batch size to 32 stabilized the training slightly (Accuracy 0.42), though still below supervised baselines.

Table 6: Sensitivity of PPO training to batch size. Small batch sizes lead to total collapse due to sparse rewards.

Batch Size	Final Accuracy
1	0.00
8	0.01
32	0.42
64	0.22

In contrast, DPO and ORPO showed much greater stability without extensive hyperparameter tuning. Figure 5 visualizes the training loss and accuracy for the DPO approach on the Single-LLM dataset. The smooth convergence of accuracy (blue line) demonstrates that the model effectively learns from the AI-generated preference pairs.

Additionally, we monitored the reward margin during training. Figure 6 shows the margin between

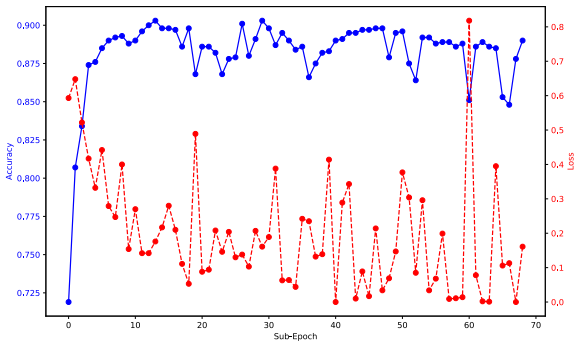


Figure 5: Training dynamics. The red dashed line represents loss (right axis) and the blue solid line represents accuracy (left axis) during fine-tuning. The model converges stably.

chosen and rejected responses. The positive trend confirms that the model is successfully optimizing for the principles defined in our RLAIF pipeline.

6.5 ORPO Results

During evaluation, both the Base SFT and ORPO models were tested using the Simple Generation zero-shot prompt. On the BIRD benchmark (Table 7), we observed a nuanced trade-off. ORPO led to a slight decrease in accuracy on "Simple" and "Moderate" queries, resulting in a minor drop in overall average accuracy (from 0.651 to 0.646). How-

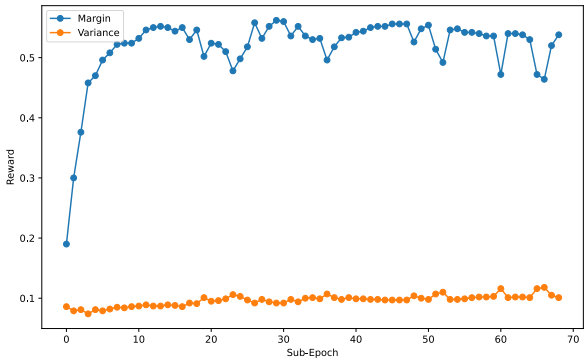


Figure 6: Evolution of the Reward Margin during training. The increasing margin indicates the model is learning to distinguish between high-quality and low-quality SQL generations.

ever, ORPO significantly improved performance on "Challenging" queries (from 44.4% to 47.2%). Thus, while monolithic preference optimization may introduce slight regression on basic syntax following, it effectively aligns the model with complex logical reasoning without suffering from the catastrophic instability observed in PPO.

Due to computational constraints, these metrics represent single-run results. Future work will evaluate variance across multiple training seeds to establish the statistical significance of the gains.

Table 7: Accuracy results on BIRD Development Set categories using ORPO.

Method	Simple	Moderate	Challenging	Average
Base SFT	0.715	0.589	0.444	0.651
ORPO	0.704	0.585	**0.472**	0.646

7 Discussion

Our findings challenge the assumption that standard RLHF methods transfer seamlessly to code generation. The sparsity of the reward signal in SQL makes PPO training unstable. ORPO offers a compelling alternative by integrating alignment into the supervised training process.

We observed that Reward Models trained on data generated by multiple LLMs (Multi-LLM) failed to generalize when scoring the Single-LLM dataset during training. This suggests that for RLAIF to function effectively, the Reward Model should be trained on the distribution of the specific model being fine-tuned, or a highly diverse set of models.

In RLHF for text generation, rewards are often dense (e.g., style, tone). In SQL, the reward is sparse and binary (executable vs. not). DPO struggles with "credit assignment"—if a model generates a wrong SQL query, DPO penalizes the entire sequence. Without Chain-of-Thought traces to indicate where the reasoning failed, the model cannot learn effectively. ORPO mitigates this by balancing the odds ratio, effectively focusing on the differentiation between valid and invalid patterns without requiring an explicit reward model.

The trade-off between complex prompting (Din-SQL) and simpler self-correction suggests that for deployment on mid-sized models (20B-34B parameters), simpler, iterative strategies are more robust. Complex chains of thought can overwhelm the attention mechanism of smaller models, leading to hallucinations of schema elements.

8 Conclusion and Future Work

This work presents a holistic view of optimizing LLMs for NL2SQL. We identified that for domain-specific applications, lightweight alignment via ORPO combined with robust self-correction prompting yields an effective balance of performance and reliability. While our computational limits restricted PPO evaluations to 7B models—where we observed severe instability due to sparse binary rewards—ORPO demonstrated stable scaling to 20B parameters, effectively improving complex logical reasoning without catastrophic collapse. Our error analysis revealed that complex Chain-of-Thought prompting can be detrimental for smaller models, inducing hallucinations. Finally, we demonstrated that RLAIF using execution and semantic principles is a promising path to alignment without expensive human annotation. While this approach champions reproducible NLP by bypassing closed-source LLM judges or costly multi-agent deployments, future work must establish the correlation between our AI-generated semantic feedback and expert human judgments to fully validate the teacher model's efficacy.

Future work will focus on scaling ORPO to larger distributed models and further refining automated reward signals for RLAIF. Specifically, we aim to explore hybrid approaches that combine the stability of ORPO with the reasoning capabilities of agentic workflows, potentially using "partial rewards" for intermediate reasoning steps to address the sparsity issue inherent in SQL generation.

References

Sihem Amer-Yahia, Jasmina Bogojeska, Roberta Facchinetti, Valeria Franceschi, Aristides Gionis, Katja Hose, Georgia Koutrika, Roger Kouyos, Matteo Lissandrini, Silviu Maniu, and 1 others. 2025. Towards reliable conversational data analytics. In *2025 EDBT/ICDT 2025 Joint Conference*.

Samuel David Arcadinho, David Aparício, Hugo Veiga, and António Alegria. 2022. T5ql: Taming language models for sql generation. In *Proceedings of the 2nd Workshop on Natural Language Generation, Evaluation, and Metrics (GEM)*, pages 276–286.

Yuntao Bai and 1 others. 2022. Constitutional ai: Harmlessness from ai feedback. *arXiv preprint arXiv:2212.08073*.

Debarun Bhattacharjya, Balaji Ganesan, Michael Glass, Junkyu Lee, Radu Marinescu, Katsiaryna Mirylenka, and Xiao Shou. 2024. Consistency-based black-box uncertainty quantification for text-to-SQL. In *Statistical Foundations of LLMs and Foundation Models (NeurIPS 2024 Workshop)*.

Debarun Bhattacharjya, Balaji Ganesan, Junkyu Lee, Radu Marinescu, Katya Mirylenka, Michael Glass, and Xiao Shou. 2025. Simba uq: Similarity-based aggregation for uncertainty quantification in large language models. In *Conference on Empirical Methods in Natural Language Processing*.

Robin Chan, Katsiaryna Mirylenka, Thomas Gschwind, Christoph Miksovic, Paolo Scotton, Enrico Toniato, and Abdel Labbi. 2024. Adapting llms for structured natural language api integration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 991–1000.

Paul F Christiano and 1 others. 2017. Deep reinforcement learning from human preferences. *NeurIPS*.

Michele Dallachiesa, Besmira Nushi, Katsiaryna Mirylenka, and Themis Palpanas. 2011. Similarity matching for uncertain time series: analytical and experimental comparison. In *Proceedings of the 2nd ACM SIGSPATIAL International Workshop on Querying and Mining Uncertain Spatio-Temporal Data*, pages 8–15.

defog ai. 2024. Sqlcoder. `https://github.com/defog-ai/sqlcoder`.

Jiwoo Hong, Noah Lee, and James Thorne. 2024. Orpo: Monolithic preference optimization without reference model. In *EMNLP*.

Vagelis Hristidis and 1 others. 2003. Efficient ir-style keyword search over relational databases. In *VLDB*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, and 1 others. 2022. Lora: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Itay Hubara, Matthieu Courbariaux, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. 2018. Quantized neural networks: Training neural networks with low precision weights and activations. *Journal of Machine Learning Research*.

George Katsogiannis-meimarakis, Katya Mirylenka, Paolo Scotton, Francesco Fusco, and Abdel Labbi. 2026. In-depth analysis of llm-based schema linking. In *International Conference on Extending Database Technology*.

Evgeny Krivosheev, Mattia Atzeni, Katsiaryna Mirylenka, Paolo Scotton, Christoph Miksovic, and Anton Zorin. 2021. Business entity matching with siamese graph convolutional networks. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 35, pages 16054–16056.

Evgeny Krivosheev, Katsiaryna Mirylenka, Mattia Atzeni, and Paolo Scotton. 2023. Graph neural networks for entity matching. In *2023 IEEE International Conference on Big Data (BigData)*, pages 6212–6214. IEEE.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *NeurIPS*.

Junyi Li, Jie Chen, Ruiyang Ren, Xiaoxue Cheng, Wayne Xin Zhao, Jian yun Nie, and Ji-Rong Wen. 2024. The dawn after the dark: An empirical study on factuality hallucination in large language models. In *Annual Meeting of the Association for Computational Linguistics*.

Raymond Li, Loubna Ben Allal, Yangtian Zi, Niklas Muennighoff, Denis Kocetkov, Chenghao Mou, Marc Marone, Christopher Akiki, Jia Li, Jenny Chim, and 1 others. 2023b. Starcoder: may the source be with you! *Trans. Mach. Learn. Res.*

Katsiaryna Mirylenka, Paolo Scotton, Christoph Adrian Miksovic Czasch, and Andreas Schade. 2021. Similarity matching systems and methods for record linkage. US Patent 11,182,395.

Mayank Mishra and 1 others. 2024. Granite code models: A family of open foundation models for code intelligence. *Preprint*, arXiv:2405.04324.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2025a. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. In *ICLR*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and 1 others. 2025b. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*.

Rafael Rafailov and 1 others. 2023. Direct preference optimization: Your language model is secretly a reward model. *NeurIPS*.

Baptiste Roziere and 1 others. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950*.

Ashish Vaswani and 1 others. 2017. Attention is all you need. In *NeurIPS*.

Bing Wang and 1 others. 2025a. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *COLING*.

Pengfei Wang, Baolin Sun, Xuemei Dong, Yaxun Dai, Hongwei Yuan, Mengdie Chu, Yingqi Gao, Xiang Qi, Peng Zhang, and Ying Yan. 2025b. Agentar-scale-sql: Advancing text-to-sql through orchestrated test-time scaling. *Preprint*, arXiv:2509.24403.

David Warren and Fernando Pereira. 1982. An efficient easily adaptable system for interpreting natural language queries. *Computational Linguistics*.

Lukas Wertz, Jasmina Bogojeska, Katsiaryna Mirylenka, and Jonas Kuhn. 2023. Reinforced active learning for low-resource, domain-specific, multi-label text classification. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 10959–10977.

Lukas Wertz, Katsiaryna Mirylenka, Jonas Kuhn, and Jasmina Bogojeska. 2022. Investigating active learning sampling strategies for extreme multi label text classification. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pages 4597–4605.

Quan Xiao, Debarun Bhattacharjya, Balaji Ganesan, Radu Marinescu, Katsiaryna Mirylenka, Nhan H. Pham, Michael Glass, and Junkyu Lee. 2025. The consistency hypothesis in uncertainty quantification for large language models. In *Proceedings of the Conference on Uncertainty in Artificial Intelligence (UAI)*.

Tao Yu and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Daniel M Ziegler and 1 others. 2019. Fine-tuning language models from human preferences. *arXiv preprint arXiv:1909.08593*.

A Constrained Decoding Error Analysis

When evaluating constrained decoding (Section 3.3), we observed an overall improvement in execution accuracy but a counter-intuitive spike in the total volume of SQL errors generated by the model. Table 8 provides a detailed quantitative breakdown of these specific error types.

Table 8: Detailed breakdown of SQL error types generated during unconstrained vs. constrained decoding on the IBM BI Dataset.

Error Type	Unconstr.	Constr.
Undefined Column	11	10
Ambiguous Column	0	8
Undefined Table	3	2
Undefined Function	1	5
Syntax Error	0	9
Division by Zero	0	1

The most significant increases occurred in Syntax Errors (9 vs 0) and Ambiguous Column errors (8 vs 0) compared to unconstrained generation. Analyzing the raw generated queries reveals that the Syntax Errors predominantly stem from queries being cut off by the maximum output length threshold.

Consider the following two examples of failed constrained generations:

```
-- Example 1: Repetition Loop
SELECT product_id, manufacturer, COUNT(
    duns_number) AS c FROM tls.inventory
GROUP BY product_id, product_id,
    product_id, product_id, product_id,
[...]
product_id, product_id, product_id,
    product_id, product

-- Example 2: Runaway WHERE Clause
SELECT tls.client_site.client_name FROM
    tls.client_site JOIN tls.inventory
ON tls.client_site.duns_number = tls.
    inventory.duns_number WHERE
tls.client_site.coverage_type = 'Data
    Center' AND
tls.client_site.coverage_name = 'Data
    Center' AND
tls.client_site.market = 'US' AND tls.
    client_site.geo = 'US'
[...]
tls.client_site.global_buying_group_id
    IS NOT NULL AND
tls.client_site.global_buying_group_name
     IS NOT NULL AND
tls.client_site.domestic_buying_group_id
     IS NOT NULL AND
```

In the first example, the constrained LLM gets trapped in a repetitive generation loop. In the second example, it appends an excessive number of unnecessary conditions to the `WHERE` clause. In both scenarios, the constraining algorithm appears to reduce the probability of the EOS token relative to other valid SQL tokens. Because our context-free

grammar strictly constrains the output syntactically but not semantically, we cannot forcefully trigger an EOS token to terminate the sequence once the logical intent has been satisfied.

Similarly, "Undefined Column" errors persist because our constrained decoding algorithm does not currently enforce that the tables for all columns mentioned in the `SELECT` clause must eventually be included in the `FROM` or `JOIN` clauses. Finally, the "Ambiguous Column" errors generated by the constrained model stem from the fact that the algorithm does not strictly enforce the `<table_name>.<column_name>` notation within `JOIN` conditions, allowing the LLM to generate ambiguous references when multiple tables share identical column names.

B Fine-tuning Hyperparameters

Specific training hyperparameters included:

- **LoRA:** Rank $r = 16$, Alpha $a = 8$, Dropout 0.05.
- **Optimizer:** AdamW with weight decay 0.1.
- **Learning Rate:** $1e - 5$.
- **Batch Size:** 16 with Gradient Accumulation Steps of 16.
- **Max Input Tokens:** 8192 (8k context window).
- **Epochs:** Models were typically trained for 1 to 3 epochs, with early stopping based on validation loss to prevent overfitting.