SPFT-SQL: Enhancing Large Language Model for Text-to-SQL Parsing by Self-Play Fine-Tuning

Anonymous ACL submission

Abstract

Despite the significant advancements of self-play fine-tuning (SPIN), which can transform a weak large language model (LLM) into a strong one through competitive interactions between models of varying capabilities, it still faces challenges in the Text-to-SQL task. SPIN does not generate new information, and the large number of correct SQL queries produced by the opponent model during self-play reduces the main model's ability to generate accurate SQL queries. To address this challenge, we propose a new self-play fine-tuning method tailored for the Text-to-SQL task, called SPFT-SQL. Prior to self-play, we introduce a validation-based iterative fine-tuning approach, which synthesizes high-quality fine-tuning data iteratively based on the database schema and validation feedback to enhance model performance, while building a model base with varying capabilities. During the self-play fine-tuning phase, we propose an error-driven loss method that incentivizes incorrect outputs from the opponent model, enabling the main model to distinguish between correct SQL and erroneous SQL generated by the opponent model, thereby improving its ability to generate correct SQL. Extensive experiments and in-depth analyses on six open-source LLMs and five widely used benchmarks demonstrate that our approach outperforms existing state-of-the-art (SOTA) methods.

1 Introduction

Text-to-SQL (Qin et al., 2022; Li et al., 2024b) aims to automatically convert natural language questions into SQL queries, enabling non-expert users to easily retrieve information from databases. Recent studies (Sun et al., 2024; Li et al., 2024a; Pourreza and Rafiei, 2024b) have demonstrated that supervised fine-tuning (SFT) (Ouyang et al., 2022) can significantly enhance performance on Text-to-SQL tasks by transforming a general-purpose open-source LLM into a specialized one. Additionally,

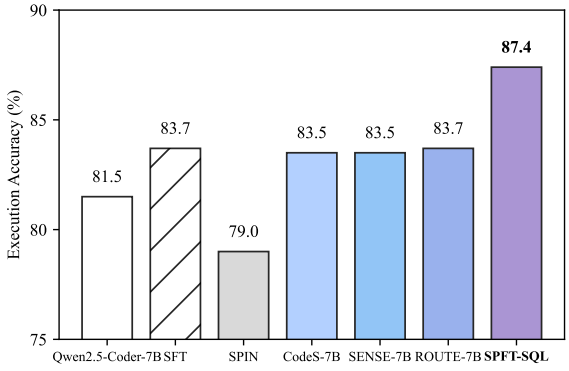

Figure 1: Comparison results on the Spider (Yu et al., 2018) dataset, the base model of SFT, SPIN (Chen et al., 2024b), and SPFT-SQL is Qwen2.5-Coder 7B.

SFT-based approaches have gained widespread research attention due to their potential to address privacy risks and reduce overhead associated with closed-source LLMs (e.g., GPT-4 (Achiam et al., 2023)) (Gao et al., 2024; Pourreza and Rafiei, 2024a; Lee et al., 2024). However, a major challenge for SFT-based methods is the high cost of acquiring Text-to-SQL data, which typically requires manual expert annotation.

To address this issue, recent efforts (Yang et al., 2024; Li et al., 2024a; Zhang et al., 2024b) have proposed data synthesis strategies for generating Text-to-SQL data and fine-tuning open-source models, yielding significant performance improvements (see Figure 1). However, these methods still rely on closed-source LLMs, such as GPT-3.5/4 (Achiam et al., 2023), for data synthesis, raising privacy concerns. In response, ROUTE (Qin et al., 2025) introduced a method for synthesizing fine-tuning data for tasks like Text-to-SQL and Schema Linking using open-source models, improving model generalization through multi-task supervised fine-tuning and achieving a new state-of-the-art (SOTA) performance. However, the limited generation capacity of open-source models restricts the quality

of synthetic data, which in turn limits model performance.

An alternative approach involves iteratively synthesizing data through self-play fine-tuning (SPIN) (Chen et al., 2024b; Cheng et al., 2025; Wu et al., 2024) to transform a weak LLM into a stronger one. Self-play, which has been successfully applied in domains such as reasoning (Cheng et al., 2025), AlphaGo (Silver et al., 2016), and AlphaZero (Silver et al., 2017), enables models to compete with themselves at various stages, enhancing both performance and data synthesis capabilities while overcoming the limitations of open-source model generation. In the context of the Text-to-SQL task, the only prior work (Liu et al., 2022) applied self-play to multi-turn Text-to-SQL, generating multiple rounds of intermediate questions and answers for data augmentation. While this method improved performance in multi-turn tasks, it is not applicable to single-turn Text-to-SQL, as it only generates intermediate data based on existing annotated pairs.

This motivates us to conduct a thorough evaluation of SPIN in the Text-to-SQL task, assessing its potential as an alternative approach. As shown in Figure 1, applying the existing SPIN method (Chen et al., 2024b) to Text-to-SQL results in a significant performance drop, which is much lower than that of SFT-based methods utilizing existing data synthesis techniques (Qin et al., 2025; Li et al., 2024a; Yang et al., 2024). A subsequent analysis of failure cases reveals two key challenges for SPIN in the Text-to-SQL domain. First, SPIN only synthesizes SQL queries from existing natural language questions, without generating new information. This limitation restricts the model's ability to improve, and repeated training leads to overfitting. Second, the self-play mechanism in SPIN treats all data generated by the opponent model as incorrect, which results in many valid SQL queries being discarded as erroneous, thus hindering the model's ability to learn from errors.

To address these challenges, we propose a self-play fine-tuning method for Text-to-SQL tasks, called SPFT-SQL. Specifically, prior to self-play, we introduce a verification-based iterative supervised fine-tuning approach that iteratively synthesizes high-quality data for fine-tuning the LLM. This method randomly selects schemas (e.g., tables and columns) from the database and combines them with SQL templates to generate executable SQL queries. Corresponding natural language questions (NLQs) are then synthesized using a SQL-to-Text model. The synthesized NLQ-SQL pairs are used to fine-tune the Text-to-SQL model, enhancing its performance. The SQL-to-Text model is subsequently updated with the synthesized data that passes a validation strategy. During self-play fine-tuning, the strongest model from the previous stage serves as the main model, while the weakest model acts as the opponent. We introduce an error-driven loss function that penalizes correct SQL queries generated by the opponent model and incentivizes the generation of incorrect queries. This mechanism enables the main model to better distinguish between correct and incorrect results, thus improving its ability to generate correct SQL queries. In the next iteration, the newly acquired main model is incorporated into the next round of supervised fine-tuning.

The main contributions of this work are as follows:

- We first evaluated the performance of the SPIN method on the Text-to-SQL task and found that the existing SPIN method performs poorly in this context. This prompted us to propose a new self-play fine-tuning method specifically designed for the Text-to-SQL task.
- We propose a validation-based iterative fine-tuning framework that synthesizes data iteratively based on the database schema and improves data quality through validation feedback, thereby continuously enhancing model performance.
- We introduce an error-driven loss that incentivizes the generation of incorrect outputs by the opponent model during the self-play fine-tuning phase. This enables the main model to distinguish between correct SQL and erroneous SQL generated by the opponent model, ultimately improving the main model's ability to generate accurate SQL queries.
- Extensive experiments on five datasets and six open-source LLMs of varying types and parameter sizes. The results demonstrate that our approach not only effectively improves model performance but also outperforms other SOTA methods based on open-source models. Furthermore, after fine-tuning with our method, small-parameter open-source models outperform methods based on large-parameter, closed-source LLM.

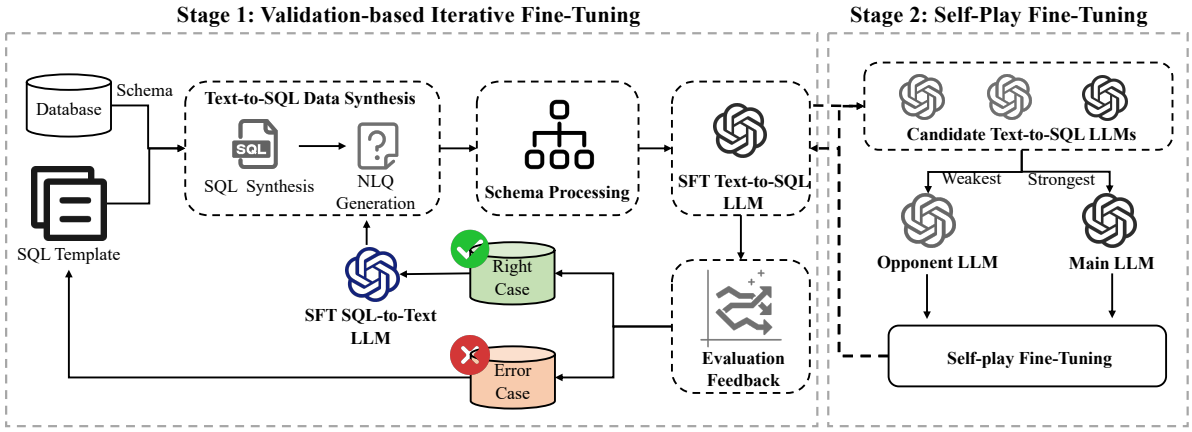

Figure 2: An overview of SPFT-SQL framework.

2 Related Works

Self-Play Fine-Tuning Self-play (Zhang et al., 2024a; DiGiovanni and Zell, 2021), where the algorithm learns by competing against itself, has gained significant attention due to its success in AlphaGo (Silver et al., 2016) and AlphaZero (Silver et al., 2017). To transform a weak LLM into a stronger one, existing studies (Chen et al., 2024b; Alami et al., 2024; Yin et al., 2024; Wu et al., 2025) have proposed introducing self-play mechanisms into LLMs without requiring additional human-annotated data. In the text-to-SQL task, there is only one prior work (Liu et al., 2022) that applies self-play to text-to-SQL. However, this method only uses self-play to generate multiple rounds of intermediate data based on existing annotated data, which makes it inapplicable to single-turn text-to-SQL tasks. In contrast to previous studies, our SPFT-SQL introduces self-play fine-tuning into the text-to-SQL task by iteratively synthesizing new text-to-SQL pairs for data augmentation. Furthermore, we propose an error-incentive loss that encourages the generation of erroneous outputs by the opponent model, thereby enhancing the main model's ability to generate correct SQL queries.

SFT-based Text-to-SQL To improve the performance of open-source LLMs on text-to-SQL tasks, existing research (Sun et al., 2024; Chen et al., 2024a; Pourreza and Rafiei, 2024b) has applied supervised fine-tuning on annotated data. However, a key challenge remains the high cost of human-annotated data. To reduce this cost, some efforts (Li et al., 2024a; Yang et al., 2024) have employed various data synthesis strategies, using LLMs to generate data for fine-tuning. However, these methods rely on the general capabilities of closed-source LLMs, such as GPT-4, which raises privacy concerns. To address this issue, Route (Qin et al., 2025) proposed a data augmentation approach to improve generalization using open-source LLMs. In contrast to previous work, our SPFT-SQL method iteratively synthesizes high-quality data through the self-play mechanism.

3 Methodology

The SPFT-SQL framework consists of two stages: Verification-Based Iterative Fine-Tuning and Self-Play Fine-Tuning, as illustrated in Figure 2. In the first stage, verification-based iterative fine-tuning continuously generates high-quality data for fine-tuning, producing various candidate text-to-SQL models for the subsequent self-play phase. During self-play fine-tuning, the strongest model from the previous stage serves as the main model, while the weakest model functions as the opponent. Using the proposed error-driven loss, self-play fine-tuning is applied between the opponent model and the main model to enhance the main model's ability to generate correct SQL queries from a NLQ. This process is iterated until the model converges.

3.1 Verification-Based Iterative Fine-Tuning

The goal of verification-based iterative fine-tuning is to generate high-quality data for fine-tuning while also providing candidate models with varying capabilities for the next phase. As shown in Figure 2, data synthesized by the Text-to-SQL synthesis module is used to fine-tune the Text-to-SQL model, incorporating schema processing. The synthetic validation data is then verified using the fine-tuned model, and the correct data is used to fine-tune the

Schema

```
CREATE TABLE people (people id number,age
number,name text,nationality text,graduation
college text,primary key (people id))…
```

SQL Template

```
SELECT DISTINCT col_text_key0 WHERE
col_number_key1 < cell_value
```

Generate SQL

```
SELECT DISTINCT nationality FROM people
WHERE age < 30
```

Generate Question

```
What are the different nationalities of
people younger than 30?
```

Figure 3: An example of Text-to-SQL data synthesis.

SQL-to-Text model, enhancing its NLQ generation ability. Data that fails validation provides SQL templates for the next iteration. The fine-tuned model is then used as a candidate LLM for the next self-play fine-tuning phase. Through this approach, we achieve positive reinforcement for the SQL-to-Text model and defect correction for the Text-to-SQL model, enabling the synthesis of higher-quality data and enhancing overall model performance.

3.1.1 Text-to-SQL Data Synthesis

The Text-to-SQL fine-tuning data synthesis process begins by generating the SQL query, followed by the synthesis of the corresponding NLQ.

SQL Synthesis To generate new SQL queries based on training data schemas, we employ a template-based approach as outlined by (Hu et al., 2023). First, a pool of SQL templates is created by normalizing schema-related mentions (columns and values) and removing JOIN phrases. A template is then sampled according to the training distribution, and tables and columns are selected with constraints to fill the normalized slots within the template.

To accurately extract the SQL template while preserving key relationships, we leverage the general understanding capabilities of the LLM. The prompt used for this extraction is defined in Appendix A.1. As shown in Figure 3, the final template maintains the query structure and data types, allowing it to adapt to various query scenarios. By omitting the `FROM` and `JOIN` clauses, the template becomes independent of specific table names, yet it retains essential query structures (e.g., `SELECT`, `WHERE`, `GROUP BY`, `HAVING`) to ensure consistency. Foreign key relationships are denoted using a special format (e.g., `col_number_key_fk`).

Once the template is generated, the method takes as input the database d and the SQL template $t = (q, c, v)$, where t consists of the query structure q, the set of columns c, and the set of values v. For columns c_1 to c_m in the template, a column is randomly selected and replaced from those that match the data types in the database. During the column selection process, if a column z comes from an already selected table, it is assigned a weight $p = 1$; otherwise, the weight is adjusted based on schema distance and accumulated through iterations, ensuring that the final column selection adheres to database schema consistency. After filling in the columns, corresponding values v_1 to v_n are retrieved and randomly filled from the database. This process leverages the database schema information to ensure that both column selection and value filling respect logical constraints and data type matching, thereby generating structurally consistent and logically sound SQL statements, as shown in Figure 3.

NLQ Synthesis To ensure that the synthesized NLQ aligns with the intent of the SQL query, we iteratively fine-tune a SQL-to-Text model to generate the corresponding NLQ based on a given SQL query. The fine-tuning SQL-NLQ pairs are derived from the correct data synthesized in the previous iteration. As self-play progresses, the model's performance improves, leading to higher-quality synthetic data and, in turn, enhanced performance of the SQL-to-Text model.

3.1.2 Schema Processing and SFT

To effectively utilize the Text-to-SQL synthetic data for fine-tuning LLMs, we address the challenge of capturing implicit patterns between database schemas and NLQs, which is complicated by the complexity of database structures. Inspired by (Li et al., 2024a), we introduce schema processing during SFT, employing three strategies: Structured Schema Extraction, Context-Aware Value Matching, and Database Metadata Augmentation.

Structured Schema Extraction filters irrelevant information by selecting the most relevant tables and columns, improving the model's focus on the database structure. Context-Aware Value Matching enhances the query-database association by aligning query columns with their corresponding values, ensuring more accurate SQL conditions. Finally, Database Metadata Augmentation incorpo-

rates metadata such as key relationships, data types, and annotations, providing richer context for understanding table relationships and field semantics. These strategies work together to progressively enhance the model's SQL generation capabilities.

3.1.3 Evaluation Feedback

The fine-tuned Text-to-SQL model is evaluated on the synthesized validation set D_{val} using an SQL executor E, and samples are classified based on execution results:

$$\begin{cases} y_+ = y', & \text{if } \mathbf{E}(y') = \mathbf{E}(y) \\ y_- = y', & \text{if } \mathbf{E}(y') \neq \mathbf{E}(y) \end{cases} \tag{1}$$

where y' represents the generated results of the fine-tuned Text-to-SQL model on D_{val}, while y refers to the ground truth SQL in D_{val}. To enhance the generalization ability of SQL-to-Text, y_+ is used as training data for the next iteration. This serves as positive feedback, boosting the model's ability to generate diverse question-answer pairs. For incorrect samples (y_-), the SQL templates from these queries are selected for the next iteration to generate new SQL-question pairs, allowing the model to correct errors. By combining these two strategies, a collaborative optimization mechanism is established, progressively reducing the proportion of incorrect samples while improving the quality of the training data. As the iterative fine-tuning continues, both the quantity and diversity of synthetic data increase, leading to improved model performance.

3.2 Self-Play Fine-Tuning

Based on the evaluation accuracy on the synthetic validation data D_{val} constructed in the previous stage, the model with the highest accuracy is selected as the main model π_m, and the model with the lowest accuracy is chosen as the opponent model π_o. The objective of self-play fine-tuning is to train the main model to distinguish between correct SQL and incorrect SQL generated by the opponent model, thereby enhancing its ability to generate accurate SQL. The detailed procedure is outlined in Algorithm 1.

In each round of self-play, we first evaluate the predictions of π_o on the validation data, obtaining y_+ and y_-, and use this data to fine-tune π_m. To enhance the main model's ability to distinguish negative samples, we propose an error-driven loss function that penalizes incorrect SQL generated by the opponent model.

Algorithm 1 Self-Play Fine-Tuning

Input: Candidate model set $\mathcal{M} = \{\pi_0, \pi_1, ..., \pi_n\}$, validation dataset $\mathcal{D}_{val}$, preference scaling β, max iterations T
Output: Optimized model π_m
for $t = 1$ to T **do**
  Model Selection:
  $\pi_m = \arg\max_{\pi \in \mathcal{M}} \text{Acc}(\pi, \mathcal{D}_{val})$
  $\pi_o = \arg\min_{\pi \in \mathcal{M}} \text{Acc}(\pi, \mathcal{D}_{val})$
  Preference Data Generation:
  Use π_o to generate preference pairs (y_+, y_-) on $\mathcal{D}_{val}$
  Model Optimization:
  Update π_m via gradient descent using Equation (2)
  Add optimized π_m to $\mathcal{M}$
end for
Return: π_m

$$\mathcal{L}_{\text{Self-Play}}(\pi_m; \pi_o) = -\mathbb{E}_{(x, y_+, y_-) \sim \mathcal{D}_{\text{val}}} \Bigg[\log \sigma \Bigg(\beta \log \frac{\pi_m(y_+|x)}{\pi_o(y_+|x)} - \beta \log \frac{\pi_m(y_-|x)}{\pi_o(y_-|x)} \Bigg) \Bigg] \tag{2}$$

where σ represents the logistic function, x is the natural language question, and β is the regularization parameter that controls the learning bias of the main model.

This loss function reinforces the main model's ability to generate correct SQL by comparing the probability differences between (π_m) and (π_o) for correct SQL (y_+) and incorrect SQL (y_-). The first term, $\log \frac{\pi_m(y_+|x)}{\pi_o(y_+|x)}$, encourages the main model to have higher confidence in generating correct SQL than the opponent model. The second term, $\log \frac{\pi_m(y_-|x)}{\pi_o(y_-|x)}$, penalizes the main model if its behavior is similar to the opponent model when generating incorrect SQL. This approach enables the main model to learn error patterns from the opponent model and avoid making similar mistakes in SQL generation.

4 Experiments

4.1 Experiment Setup

Benchmarks To evaluate the effectiveness of our approach, we conduct experiments on five Text-to-SQL benchmarks, including the widely

Methods	SPIDER			SPIDER-Variants					BIRD	
	Dev		Test	Syn		Realistic		DK	Dev	
	EX	TS	EX	EX	TS	EX	TS	EX	EX	VES
Prompting with GPT										
GPT-4(Achiam et al., 2023)	72.9	64.9	76.1	-	-	-	-	-	46.4	49.8
DIN-SQL+GPT4(Pourreza and Rafiei, 2024a)	82.8	74.2	85.3	-	-	-	-	-	50.7	58.8
MAC-SQL+GPT4(Wang et al., 2023)	86.8	-	82.8	-	-	-	-	-	59.4	66.2
DAIL-SQL+GPT4(Gao et al., 2024)	83.5	76.2	86.6	-	-	-	-	-	54.8	56.1
MCS-SQL+GPT4(Lee et al., 2024)	89.5	-	89.6	-	-	-	-	-	63.4	64.8
Open-Source LLMs										
Llama3-8B(Touvron et al., 2023)	72.3	63.9	69.6	60.3	51.2	62.0	50.4	57.4	39.2	43.3
Deepseek-7B(Guo et al., 2024)	67.0	57.7	69.4	55.3	46.0	57.7	45.9	55.3	40.1	44.5
Qwen2.5 Coder-1.5B(Hui et al., 2024)	72.4	62.5	72.3	55.7	45.4	60.4	46.5	62.0	40.6	43.3
Qwen2.5 Coder-7B(Hui et al., 2024)	83.5	79.2	81.5	69.8	64.2	75.4	70.9	68.0	51.5	55.3
Qwen2.5 Coder-14B(Hui et al., 2024)	83.8	78.0	84.9	74.3	66.8	76.4	69.1	69.7	58.0	62.8
Qwen2.5 Coder-32B(Hui et al., 2024)	79.6	73.9	82.3	73.7	67.6	75.6	68.3	71.0	58.1	61.7
Fine-Tuning with Open-Source LLMs										
Codes-7B+SFT(Li et al., 2024a)	85.4	80.3	83.5	76.9	70.0	82.9	77.2	72.0	57.2	58.8
Codes-15B+SFT(Li et al., 2024a)	84.9	79.4	85.0	77.0	67.4	83.1	75.6	70.7	58.5	56.7
ROUTE+Llama3-8B(Qin et al., 2025)	86.0	80.3	83.9	77.4	70.2	80.9	72.6	74.6	57.3	60.1
ROUTE+Qwen2.5-7B(Qin et al., 2025)	83.6	77.5	83.7	-	-	-	-	-	55.9	57.4
ROUTE+Qwen2.5-14B(Qin et al., 2025)	87.3	80.9	87.1	-	-	-	-	-	60.9	65.2
DTS-SQL+Deepseek 7B (Pourreza and Rafiei, 2024b)	85.5	-	84.4	-	-	-	-	-	55.8	60.3
SENSE-7B(Yin and Neubig, 2017)	83.2	81.7	83.5	72.6	64.9	82.7	75.6	77.9	51.8	-
SENSE-13B(Yin and Neubig, 2017)	84.1	**83.5**	86.6	77.6	70.2	84.1	76.6	**80.2**	55.5	-
Llama3-8B+SFT(Touvron et al., 2023)	79.5	73.6	80.9	66.4	60.1	71.1	62.8	61.7	51.8	55.3
Deepseek-7B+SFT(Guo et al., 2024)	78.6	71.9	81.5	64.8	57.5	69.3	61.4	60.7	53.9	57.1
Qwen2.5 Coder-1.5B+SFT(Hui et al., 2024)	76.8	70.2	78.0	59.0	51.6	64.8	56.3	60.9	44.3	45.6
Qwen2.5 Coder-7B+SFT(Hui et al., 2024)	82.9	79.0	83.7	68.3	62.3	75.2	69.5	66.5	54.4	56.1
Qwen2.5 Coder-14B+SFT(Hui et al., 2024)	84.8	79.6	84.4	68.4	62.1	74.4	66.7	70.3	58.5	63.9
Qwen2.5 Coder-32B+SFT(Hui et al., 2024)	85.2	79.5	86.4	77.2	71.1	76.0	70.1	72.5	61.2	66.6
Self-Play Method										
SPIN+Llama3-8B(Chen et al., 2024b)	79.8	73.6	80.2	66.8	60.2	69.3	60.0	61.9	32.4	37.3
SPIN+Deepseek-7B(Chen et al., 2024b)	61.8	51.5	63.9	45.6	35.8	52.2	37.6	48.2	27.0	29.1
SPIN+Qwen2.5 Coder-1.5B(Chen et al., 2024b)	67.6	60.5	68.5	55.0	46.4	57.2	45.5	54.0	21.0	22.1
SPIN+Qwen2.5 Coder-7B(Chen et al., 2024b)	78.0	73.1	79.0	71.2	57.5	63.6	65.9	61.8	34.8	37.7
SPIN+Qwen2.5 Coder-14B(Chen et al., 2024b)	82.3	76.9	81.4	72.8	62.0	68.1	66.1	67.8	36.8	41.2
SPFT-SQL+Llama3-8B	83.0	75.4	86.4	76.1	69.1	85.0	**77.6**	74.2	60.6	65.4
SPFT-SQL+Deepseek-7B	82.3	78.0	86.0	76.5	70.0	80.3	74.8	72.5	58.3	64.0
SPFT-SQL+Qwen2.5 Coder-1.5B	79.7	73.5	82.3	66.7	59.4	75.4	67.3	67.3	54.0	59.9
SPFT-SQL+Qwen2.5 Coder-7B	87.2	81.3	87.4	75.1	67.6	83.3	75.6	75.5	61.0	67.0
SPFT-SQL+Qwen2.5 Coder-14B	87.7	81.9	89.0	78.4	71.3	84.6	**77.6**	77.0	63.6	68.9
SPFT-SQL+Qwen2.5 Coder-32B	**87.8**	83.2	**89.1**	**81.7**	**72.3**	**86.2**	76.8	75.5	**65.2**	**70.5**

Table 1: Performance of different methods on SPIDER, BIRD, and SPIDER-variants Datasets.

used cross-domain datasets SPIDER (Yu et al., 2018) and BIRD (Li et al., 2024b), along with three SPIDER-derived versions: SPIDER-SYN (Gan et al., 2021a), SPIDER-Realistic (Deng et al., 2021), and SPIDER-DK (Gan et al., 2021b). SPIDER contains 7,000 training samples, 1,034 development samples, and 2,147 test samples, covering 206 databases across 138 domains. BIRD introduces more complex domain-specific queries, comprising 12,751 question-SQL pairs from 37 domains, including finance, healthcare, and education. SPIDER-SYN enhances 20 databases in the SPIDER validation set through synonym substitution, SPIDER-Realistic extracts SQL from 19 databases to generate more realistic questions, and SPIDER-DK introduces 535 knowledge-augmented queries across six databases to improve domain reasoning.

Evaluation Metrics We evaluate model performance using execution accuracy (EX)(Yu et al., 2018) and test-suite accuracy (TS)(Zhong et al., 2020) on SPIDER and its variants. EX measures if the generated SQL matches the gold SQL execution, while TS verifies its performance across multiple test cases with database augmentation. For BIRD, following its official settings, we report EX and Valid Efficiency Score (VES)(Li et al., 2024b), which measures SQL execution efficiency.

Models We evaluate the generalizability of our method using six open-source LLMs, including Llama3-8B-Instruct (Touvron et al., 2023), Deepseek-Coder-7B-Instruct (Guo et al., 2024), and Qwen2.5-Coder (Hui et al., 2024) in sizes 1.5B, 7B, 14B, and 32B.

Baselines Our baselines are divided into four categories: prompting methods with GPT-4, prompting methods with open-source LLMs, fine-tuning-based methods with open-source LLMs, and self-play-based methods with open-source LLMs. The first group includes DIN-SQL (Pourreza and Rafiei, 2024a), MAC-SQL (Wang et al., 2023), DAIL-SQL (Gao et al., 2024), MCS-SQL (Lee et al., 2024), and the closed-source LLM GPT-4 (Achiam et al., 2023). For the open-source LLM baselines, we use the six LLMs mentioned earlier in a zero-shot setting. The fine-tuning-based baselines are represented by specialized LLMs, including CodeS (Li et al., 2024a), ROUTE (Qin et al., 2025), DTS-SQL (Pourreza and Rafiei, 2024b), SENSE (Yin and Neubig, 2017), as well as the six base LLMs fine-tuned on five training sets using the Llama-Factory (Zheng et al., 2024) framework. Finally, for the self-play-based methods, we compare with SPIN (Chen et al., 2024b) on the five base LLMs, as the 32B model could not be used for SPIN due to resource limitations.For fairness, we reproduce several baselines using open-source repositories and conduct rigorous evaluations.

4.2 Comparison Results

Table 1 presents the performance of our method and baselines across various datasets, including the SPIDER development and test sets, BIRD development set, SPIDER-SYN, SPIDER-Realistic, and SPIDER-DK. Due to time constraints, we were unable to provide results for our SPFT-SQL on the BIRD test set. In nearly all cases, our SPFT-SQL achieves the best performance. From the results, it is evident that fine-tuning-based methods significantly improve the performance of open-source LLMs. Notably, specialized LLMs (e.g., ROUTE, SENSE, CodeS) fine-tuned on synthetic data outperform those fine-tuned on the original training set but still do not match the performance of closed-source LLMs (e.g., GPT-4). This indicates that the quality of synthetic data remains a limiting factor. The accuracy of the existing self-play method, SPIN, is not only lower than fine-tuning-based methods but also below that of the original model.

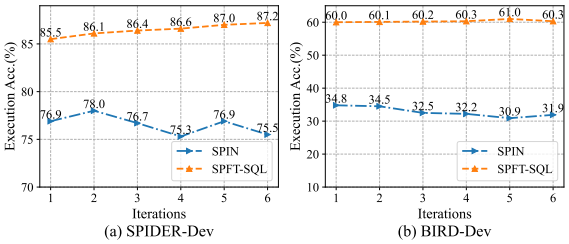

Figure 4: Comparison of Different Iteration

This is due to SPIN's failure to generate new data for the text-to-SQL task, leading to overfitting from repeated fine-tuning. In contrast, our SPFT-SQL iteratively synthesizes new data during fine-tuning and enhances data quality through a verification feedback mechanism, thereby improving model performance. As a result, our method not only improves the performance of self-play on text-to-SQL tasks but also surpasses specialized LLMs fine-tuned on synthetic data. Particularly on the SPIDER test set, our SPFT-SQL outperforms several existing prompting-based methods (e.g., DIN-SQL, DAIL-SQL, and MAC-SQL), achieving an EX score of 89.1%, significantly narrowing the gap with GPT-4-based methods.

4.3 Parameter Study

Figure 4 shows the performance of SPIN and SPFT-SQL across different iteration rounds on the SPIDER and BIRD development sets, using the Qwen2.5 Coder-7B model as the base model. Our findings reveal that as the number of training iterations increases, the accuracy of SPIN continuously decreases, indicating that SPIN suffers from overfitting with more iterations. In contrast, our method shows a steady improvement in accuracy as iterations progress, suggesting that it continues to generate high-quality data, enhancing model performance and gradually converging. Detailed experimental results for the other datasets can be found in Appendix A.3.

4.4 Synthetic Data Study

Synthetic Data Quantity As shown in Figure 5, we evaluated the impact of varying amounts of synthetic data on model performance at each round during the Verification-Based Iterative Fine-Tuning phase, using the Qwen2.5 Coder-7B model. The results indicate that generating 3,000 data records yields the best performance. This suggests that selecting the appropriate amount of synthetic data during training involves a trade-off: too little data

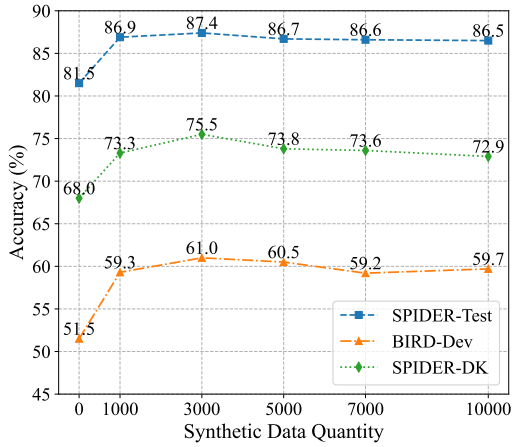

Figure 5: Performance on varing number of synthetic data each round.

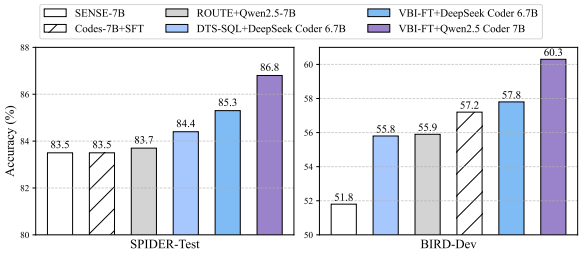

Figure 6: Performance of LLMs fine-tuned on synthetic data generated by SPFT-SQL and baselines.

	SPIDER			Bird-Dev	
	Dev-EX	Dev-TS	Test-EX	EX	VES
Qwen2.5 Coder-7B	83.5	79.2	81.5	51.5	55.3
SPFT-SQL	87.2	81.3	87.4	61.0	67.0
w/o VBI-FT	83.9	79.3	83.6	54.2	57.5
w/o Self-Play	86.6	80.4	86.8	60.3	65.3

Table 2: The ablation study results on the SPIDER and the BIRD datasets.

may result in undertraining, while generating excessive data could incur unnecessary time costs.

Synthetic Data Quality Figure 6 presents the performance of LLMs fine-tuned on synthetic data generated by our Verification-Based Iterative Fine-Tuning (VBI-FT) phase, compared to other methods, including CodeS, ROUTE, SENSE, and DTS-SQL, on the SPIDER test set and BIRD development set. The results demonstrate that SPFT-SQL significantly improves model performance by synthesizing higher-quality data. Using the same base model, SPFT-SQL boosts performance by 3.1% and 0.9% on the SPIDER test set compared to ROUTE and DTS-SQL, and by 4.4% and 2% on the BIRD development set. Compared to SENSE and CodeS, SPFT-SQL shows notable improvements ranging from 3.3% to 8.4%. This improvement can be attributed to the fact that methods like ROUTE and DTS-SQL rely on basic open-source models for data synthesis, which limits the quality of generated data due to the models' inherent capabilities. In contrast, SPFT-SQL overcomes these limitations by leveraging an iterative evaluation feedback mechanism, enhancing the quality of synthetic data and, consequently, the performance of the model.

4.5 Ablation Study

As shown in Table 2, we conducted an ablation study on the SPIDER and BIRD datasets using Qwen2.5 Coder-7B as the base LLM. The results reveal two key findings. First, Verification-Based Iterative Fine-Tuning (VBI-FT) improved performance by 3.3% to 3.8% in EX on the SPIDER dataset and by 6.8% on the BIRD dataset, highlighting its significance in enhancing core SQL synthesis abilities. Additionally, the self-play fine-tuning process resulted in an accuracy boost of 0.6% to 0.8% on both datasets, demonstrating that self-play enables the model to leverage its intrinsic capabilities without external supervision. Together, these findings underscore the effectiveness of both Verification-Based Iterative Fine-Tuning and self-play fine-tuning in improving model performance on text-to-SQL tasks. Additional results for the SPIDER-variant can be found in Appendix A.4.

5 Conclusion

In this paper, we propose a novel self-play fine-tuning framework for text-to-SQL tasks, called SPFT-SQL. SPFT-SQL enhances the capabilities of open-source models through verification-based iterative fine-tuning to generate high-quality data augmentation, while further improving the models' ability to generate accurate SQL via error-driven adversarial training in self-play scenarios. Our work represents the first effective implementation of the self-play method in text-to-SQL tasks, significantly narrowing the performance gap between open-source and closed-source models. Future research will focus on exploring cross-domain generalization capabilities and developing efficient adversarial architectures.

Limitations

Although our method has shown promising performance and significant progress across various aspects, there are several limitations and areas for further improvement. Firstly, due to resource constraints, we were unable to fine-tune the model on larger architectures, such as a 70B-parameter model. Secondly, the introduction of data synthesis and additional evaluation steps increased the computational time; while SFT training for a 7B model took approximately one hour on the resource configuration of 8*A800 (80G), our method required about two hours per training iteration under the same conditions. Nonetheless, our approach still outperforms SPIN, which requires six hours for each training round on similar resources.

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, et al. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Reda Alami, Abdalgader Abubaker, Mastane Achab, Mohamed El Amine Seddik, and Salem Lahlou. 2024. Investigating regularization of self-play language models. *arXiv preprint arXiv:2404.04291*.

Xiaojun Chen, Tianle Wang, Tianhao Qiu, Jianbin Qin, and Min Yang. 2024a. Open-sql framework: Enhancing text-to-sql on open-source large language models. *arXiv preprint arXiv:2405.06674*.

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. 2024b. Self-play fine-tuning converts weak language models to strong language models. In *Forty-first International Conference on Machine Learning*.

Pengyu Cheng, Tianhao Hu, Han Xu, Zhisong Zhang, Yong Dai, Lei Han, Xiaolong Li, et al. 2025. Self-playing adversarial language game enhances llm reasoning. *Advances in Neural Information Processing Systems*, 37:126515–126543.

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.

Anthony DiGiovanni and Ethan C Zell. 2021. Survey of self-play in reinforcement learning. *arXiv preprint arXiv:2107.02850*.

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.

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*.

Yiqun Hu, Yiyun Zhao, Jiarong Jiang, Wuwei Lan, Henghui Zhu, Anuj Chauhan, Alexander Hanbo Li, Lin Pan, Jun Wang, Chung-Wei Hang, et al. 2023. Importance of synthesizing high-quality data for text-to-sql parsing. In *Findings of the Association for Computational Linguistics: ACL 2023*, pages 1327–1343.

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*.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *arXiv preprint arXiv:2405.07467*.

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.

Qi Liu, Zihuiwen Ye, Tao Yu, Phil Blunsom, and Linfeng Song. 2022. Augmenting multi-turn text-to-sql datasets with self-play. In *The 2022 Conference on Empirical Methods in Natural Language Processing*.

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 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. In *EMNLP (Findings)*, pages 8212–8220.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, et al. 2022. A survey on text-to-sql parsing: Concepts, methods, and future directions. *arXiv preprint arXiv:2208.13629*.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2025. ROUTE: Robust multitask tuning and collaboration for text-to-SQL. In *The Thirteenth International Conference on Learning Representations*.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. 2016. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484–489.

David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, et al. 2017. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815*.

Ruoxi Sun, Sercan O Arik, Alexandre Muzio, Lesly Miculicich, Satya Kesav Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. SQL-paLM: Improved large language model adaptation for text-to-SQL. *Transactions on Machine Learning Research*.

Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. 2023. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. Mac-sql: Multi-agent collaboration for text-to-sql. *arXiv preprint arXiv:2312.11242*.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2024. Self-play preference optimization for language model alignment. *arXiv preprint arXiv:2405.00675*.

Yue Wu, Zhiqing Sun, Huizhuo Yuan, Kaixuan Ji, Yiming Yang, and Quanquan Gu. 2025. Self-play preference optimization for language model alignment. In *The Thirteenth International Conference on Learning Representations*.

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

Pengcheng Yin and Graham Neubig. 2017. A syntactic neural model for general-purpose code generation. *arXiv preprint arXiv:1704.01696*.

Yueqin Yin, Zhendong Wang, Yujia Xie, Weizhu Chen, and Mingyuan Zhou. 2024. Self-augmented preference optimization: Off-policy paradigms for language model alignment. *arXiv preprint arXiv:2405.20830*.

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 *2018 Conference on Empirical Methods in Natural Language Processing, EMNLP 2018*, pages 3911–3921. Association for Computational Linguistics.

Ruize Zhang, Zelai Xu, Chengdong Ma, Chao Yu, Wei-Wei Tu, Shiyu Huang, Deheng Ye, Wenbo Ding, Yaodong Yang, and Yu Wang. 2024a. A survey on self-play methods in reinforcement learning. *arXiv preprint arXiv:2408.01072*.

Yi Zhang, Jan Milan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2024b. Sciencebenchmark: a complex real-world benchmark for evaluating natural language to sql systems. *Proceedings of the VLDB Endowment*, 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. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 3: System Demonstrations)*, Bangkok, Thailand. Association for Computational Linguistics.

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

A Appendix

A.1 Prompt

In this section, we provide the prompts employed for the methodology described under Template Extraction, which are depicted in Figure 7.

Task

Given the following SQL query, generate an SQL template by removing the "FROM" and "JOIN" sections, ensuring that no table names appear in the template. All other parts of the query, including SELECT, WHERE, GROUP BY, and HAVING, should remain unchanged. For placeholders, use the following formats:

- **col_number_key#** for numeric columns,
- **col_text_key#** for textual columns, and
- **cell_value** for constant values.

Important Note

If two columns in the SQL query satisfy a foreign key relationship (i.e., one column is a foreign key referencing another table's primary key), explicitly indicate this relationship in the template using the placeholder format **col_number_key#_fk#**, where **fk#** represents the foreign key reference. For example, if **column_A** is a foreign key referencing **column_B**, replace **column_A** with **col_number_key0_fk1**.

Now, apply the same transformation to the SQL query below and please keep FROM and JOIN sections removed:

Input:

{"sql": "{Input SQL}"}

Schema:

{Database Schema}

Output:

Figure 7: Prompt for extracting standardized SQL templates.

A.2 Comparison with different hardness

To comprehensively evaluate the model performance, we adopted methodologies from pertinent studies (Pourreza and Rafiei, 2024a; Gao et al., 2024; Qin et al., 2025) and computed the EX score on the development sets of SPIDER and BIRD. The results presented in Table 3 and Table 4 demonstrate that the SPFT-SQL approach excels both in overall performance and across various difficulty levels, thereby further validating the efficacy of our proposed method.

A.3 Comparison of Synthetic Data Quantity

Table 5 presents the experimental results across all datasets for different amounts of synthetic data.The experimental results show that the model achieves the best performance when generating 3,000 synthetic data records. Specifically, it achieves an accuracy of 87.4% on the SPIDER-Test dataset, 87.2% on the SPIDER-Dev dataset, 61.0% on the BIRD-Dev dataset. These results indicate that generating an appropriate amount of synthetic data is crucial for improving model performance.

When the amount of generated data is relatively small, the model's performance improves but does not reach its optimal state. For example, the accuracy on the SPIDER-Realistic dataset is 82.1%, on the SPIDER-DK dataset is 73.3%, and on the BIRD-Dev dataset is 59.3%. This suggests that insufficient data may prevent the model from learning enough information, thereby limiting its performance.

On the other hand, when the amount of generated data is excessive, the model's performance declines. For instance, when generating 5,000 records, the accuracy on the SPIDER-Test dataset drops to 86.7%, on the SPIDER-Syn dataset to 72.1%, and on the BIRD-Dev dataset to 60.5%. This could incur unnecessary time costs.

In conclusion, generating 3,000 synthetic data records is an ideal choice, as it ensures data quality while maximizing model performance improvement. This finding emphasizes the importance of selecting an appropriate amount of synthetic data during training to avoid compromising the model's final performance due to insufficient or excessive data.

A.4 Ablation Study Results

Table 6 delineates the ablation study results across all datasets, shedding light on the individual contributions of various components to the overall system

Method	Easy	Medium	Hard	Extra	All
Prompting with GPT					
DIN-SQL+GPT4(Pourreza and Rafiei, 2024a)	92.3	87.4	76.4	62.7	82.8
DAIL-SQL+GPT4(Gao et al., 2024)	91.5	90.1	75.3	62.7	83.6
MCS-SQL+GPT4(Lee et al., 2024)	94.0	93.5	88.5	72.9	89.5
Fine-Tuning with Open-Source LLMs					
Codes-7B+SFT(Li et al., 2024a)	94.8	91.0	75.3	66.9	85.4
Codes-15B+SFT(Li et al., 2024a)	95.6	90.4	78.2	61.4	84.9
SENSE-7B(Yin and Neubig, 2017)	95.2	88.6	75.9	60.3	83.5
ROUTE+Qwen2.5-7B(Qin et al., 2025)	92.8	89.7	77.0	60.2	83.6
ROUTE+Qwen2.5-14B(Qin et al., 2025)	94.0	93.0	81.6	**68.1**	87.3
Self-Play Method					
SPIN+Qwen2.5 Coder-14B(Chen et al., 2024b)	91.5	87.7	74.7	63.3	82.3
SPFT-SQL+Qwen2.5 Coder-1.5B	92.3	83.4	72.4	58.4	79.7
SPFT-SQL+Qwen2.5 Coder-7B	96.4	91.9	**85.1**	62.7	87.2
SPFT-SQL+Qwen2.5 Coder-14B	95.6	**94.4**	81.6	64.5	87.7
SPFT-SQL+Qwen2.5 Coder-32B	**96.4**	93.5	80.5	67.5	**87.8**

Table 3: The performance (EX) comparison with different hardness on the SPIDER-Dev

Method	Simple	Moderate	Challenging	All
Prompting with GPT				
MAC-SQL+GPT4(Wang et al., 2023)	65.7	52.7	40.3	59.4
MCS-SQL+GPT4(Lee et al., 2024)	70.4	53.1	51.4	63.4
Fine-Tuning with Open-Source LLMs				
Codes-7B+SFT(Li et al., 2024a)	64.6	46.9	40.3	57.2
Codes-15B+SFT(Li et al., 2024a)	65.8	48.8	42.4	58.5
ROUTE+Qwen2.5-7B(Qin et al., 2025)	63.8	45.4	39.6	55.9
ROUTE+Qwen2.5-14B(Qin et al., 2025)	67.7	53.1	42.4	60.9
Self-Play Method				
SPIN+Qwen2.5 Coder-14B(Chen et al., 2024b)	45.8	24.1	20.1	36.8
SPFT-SQL+Qwen2.5 Coder-1.5B	61.1	46.5	33.3	54.0
SPFT-SQL+Qwen2.5 Coder-7B	68.7	51.6	41.7	61.0
SPFT-SQL+Qwen2.5 Coder-14B	68.8	**57.6**	49.3	63.6
SPFT-SQL+Qwen2.5 Coder-32B	**71.2**	57.4	**51.4**	**65.2**

Table 4: The performance (EX) comparison with different hardness on the BIRD-Dev

performance. The implementation of Verification-Based Iterative Fine-Tuning (VBI-FT) significantly enhanced the model's performance, with improvements ranging from 5.2% to 8.3% in EX on the SPIDER-Variants, highlighting its critical role in advancing core SQL synthesis capabilities. Furthermore, the self-play fine-tuning process contributed to an accuracy increase of 0.3% to 1.7% on the SPIDER-Variants, illustrating how self-play allows the model to optimize its inherent potential without relying on external supervision. Collectively, these results underscore the efficacy of both Verification-Based Iterative Fine-Tuning and self-play fine-tuning in boosting the model's performance on SQL synthesis tasks.

A.5 Comparison of Generated SQLs from Different Methods

To better illustrate the improvements of our method over others, we selected two examples from the SPIDER and BIRD datasets, as shown in Tables 7 and 8. In the first example, both the models fine-

Quantity	SPIDER			SPIDER-Variants					BIRD	
	Dev		Test	Syn		Realistic		DK	Dev	
	EX	TS	EX	EX	TS	EX	TS	EX	EX	VES
0	83.5	79.2	81.5	69.8	64.2	75.4	70.9	68.0	51.5	55.3
1000	86.5	80.9	86.9	**76.8**	**69.6**	82.1	**76.6**	73.3	59.3	62.7
3000	**87.2**	**81.3**	**87.4**	75.1	67.6	**83.3**	75.6	**75.5**	**61.0**	**67.0**
5000	86.3	79.6	86.7	72.1	64.3	82.8	75.6	73.8	60.5	66.2
7000	86.6	80.9	86.6	71.8	64.1	82.1	73.6	73.6	59.2	65.8
10000	86.0	80.4	86.5	72.4	64.6	82.5	73.6	72.9	59.7	62.9

Table 5: Comparison of Synthetic Data Quantity

	SPIDER			SPIDER-Variants					BIRD	
	Dev		Test	Syn		Realistic		DK	Dev	
	EX	TS	EX	EX	TS	EX	TS	EX	EX	VES
Qwen2.5 Coder-7B	83.5	79.2	81.5	69.8	64.2	75.4	70.9	68.0	51.5	55.3
SPFT-SQL	87.2	81.3	87.4	75.1	67.6	83.3	75.6	75.5	61.0	67.0
w/o VBI-FT	83.9	79.3	83.6	69.9	64.6	75.0	70.5	70.3	54.2	57.5
w/o self-play	86.6	80.4	86.8	74.8	66.8	82.7	74.4	73.8	60.3	65.3

Table 6: Ablation Study Results

tuned with SFT and SPIN lost the ability to compute ratios, while our method was able to correctly calculate the formula and select the correct fields. This is attributed to our VBI-FT method, which generates high-quality training data during the iterative process, and the guidance provided during the Self-Play phase, helping the model select the correct fields and generate accurate SQL queries. In the second example, both SFT and SPFT achieved the correct results, while SPIN produced incorrect values, further demonstrating the issue with the SPIN self-play mechanism, where the model's historical outputs are treated as negative samples.

Table 7: Comparison of Generated SQLs from Different Methods on the BIRD Dataset

Stage	Details
Data	**Question**: What is the eligible free or reduced price meal rate for the top 5 schools in grades 1-12 with the highest free or reduced price meal count of the schools with the ownership code 66? **Gold SQL**: SELECT CAST(T1.‘FRPM Count (K-12)‘ AS REAL) / T1.‘Enrollment (K-12)‘ FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T2.SOC = 66 ORDER BY T1.‘FRPM Count (K-12)‘ DESC LIMIT 5
Original Model	**Pred SQL**: SELECT frpm.CDSCode, frpm.School Name, frpm.FRPM Count (K-12), frpm.Enrollment (K-12), (frpm.FRPM Count (K-12) / frpm.Enrollment (K-12)) AS Eligible_FRPM_Rate FROM frpm JOIN schools ON frpm.CDSCode = schools.CDSCode WHERE schools.OwnershipCode = 66 ORDER BY frpm.FRPM Count (K-12) DESC LIMIT 5; **Result**: Wrong
SFT	**Pred SQL**: SELECT T2.‘Percent (%) Eligible Free (K-12)‘, T2.‘Percent (%) Eligible FRPM (K-12)‘ FROM schools AS T1 INNER JOIN frpm AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.SOC = 66 ORDER BY T2.‘FRPM Count (K-12)‘ DESC LIMIT 5; **Result**: Wrong
SPIN	**Pred SQL**: SELECT "Percent (%) Eligible Free (K-12)", "Percent (%) Eligible FRPM (K-12)" FROM frpm WHERE "2013-14 CALPADS Fall 1 Certification Status" = 66 ORDER BY "FRPM Count (K-12)" DESC LIMIT 5; **Result**: Wrong
SPFT-SQL	**Pred SQL**: SELECT cast(frpm.‘frpm count (k-12)‘ AS REAL) / frpm.‘enrollment (k-12)‘ FROM frpm INNER JOIN schools ON frpm.cdscode = schools.cdscode WHERE schools.soc = 66 ORDER BY frpm.‘frpm count (k-12)‘ DESC LIMIT 5; **Result**: Right

Table 8: Comparison of Generated SQLs from Different Methods on the SPIDER Dataset

Stage	Details
Data	**Question**: Count the number of searches made by buyers. **Gold SQL**: SELECT count(*) FROM Users AS T1 JOIN User_Searches AS T2 ON T1.user_id = T2.user_id WHERE T1.is_buyer = 1;
Original Model	**Pred SQL**: SELECT COUNT(us.search_seq) FROM Users u JOIN User_Searches us ON u.user_id = us.user_id WHERE u.is_buyer = 'Yes'; **Result**: Wrong
SFT	**Pred SQL**: SELECT count(*) FROM users AS T1 JOIN user_searches AS T2 ON T1.user_id = T2.user_id WHERE T1.is_buyer = '1' **Result**: Right
SPIN	**Pred SQL**: SELECT count(*) FROM users AS t1 JOIN user_searches AS t2 ON t1.user_id = t2.user_id WHERE t1.is_buyer = 't' **Result**: Wrong
SPFT-SQL	**Pred SQL**: SELECT count(*) FROM users JOIN user_searches ON users.user_id = user_searches.user_id WHERE users.is_buyer = 1; **Result**: Right