TALENT: Tree-structured Adaptive Learning for Efficient Text-to-SQL Generation

Anonymous ACL submission

Abstract

Text-to-SQL systems face increasing challenges in managing complex query generation tasks while maintaining computational efficiency. While recent approaches leverage Large Language Models (LLMs) through chain-based decomposition, they often struggle with error propagation and limited adaptability. To navigate these challenges, we propose TALENT, a hybrid framework that addresses these limitations through tree-structured task planning and reinforcement learning optimization. Our solution contributes two methodological advancements: (1) a flexible tree-based decomposition framework that enables targeted error recovery and reduces inter-task coupling, and (2) a reinforcement learning-enhanced adaptive path optimization mechanism that leverages historical execution patterns to enhance model performance. Our empirical evaluation, conducted on the Spider benchmark demonstrate TALENT's effectiveness, achieving 85.8% execution accuracy with minimal training examples. Through systematic ablation studies and artifact analysis, we further demonstrate the framework's enhanced robustness against dataset biases. These results indicate that structured task orchestration coupled with self-improving optimization can effectively address the demands for accuracy and reliability of text-to-SQL conversion. The complete implementation is available at https://github.com/FIC/TALENT.

1 Introduction

Text-to-SQL systems have become essential in modern human-computer interaction. While conventional end-to-end neural architectures have shown success in direct translation (Shi et al., 2018), they face increasing challenges as databases grow in complexity: complex syntax interpretation, task complexity management, and semantic disambiguation (Liu et al., 2023).

The emergence of Large Language Models (LLMs) has introduced a paradigm shift in addressing these challenges through task decomposition. Recent approaches like DINSQL (Pourreza and Rafiei, 2024) and C3SQL (Dong et al., 2023) have shown promise using the chain-based decomposition approach (Chain of Thought (Wei et al., 2022)). They face two fundamental challenges: error propagation in tightly coupled sub-tasks and limited adaptive capabilities (Suzgun et al., 2022). Their tightly coupled sequential structure means early-stage errors can propagate through the entire chain, while the interdependence between sub-tasks complicates error recovery.

We identify two distinct types of failures: execution errors (e.g., invalid SQL syntax) and planning errors (suboptimal task decomposition). While Execution errors provide clear signals for correction, planning errors often remain undetected until irreparable downstream failures occur.

Existing methods, such as ReACT (Yao et al., 2023) and Reflexion (Shinn et al., 2024), tackle this problem through global re-execution and re-planning. However, this resource-intensive approach yields diminishing returns, with each iteration consuming more resources.

Building upon these insights, we propose TALENT, a hybrid system guided by two principles: (1) Task decoupling to minimize error propagation and (2) Information augmentation through execution histories. Our key contributions include:

- A tree-structured task planning framework that enables flexible decomposition and targeted error recovery to address chain-based approaches' limitations.
- A Planning Enhancement Model that integrates curriculum-driven reinforcement learning. Using tree-structured decomposition patterns and execution feedback, developing path optimization skills.

Extensive evaluation on the Spider (Yu et al.,

2018) benchmark demonstrates TALENT's effectiveness in addressing core text-to-SQL challenges: achieving state-of-the-art performance with simple random one-shot prompting, surpassing existing In-Context-Learning Enhanced systems, including C3SQL, DINSQL, and DAILSQL. These results validate our approach's effectiveness in addressing the core challenges of text-to-SQL translation while enhancing robustness.

2 Related Work

The evolution of data access tools presents a trade-off between usability and functionality. While form-based interfaces prioritize accessibility, SQL offers comprehensive query capabilities at the cost of significant learning barriers (Codd, 1974). Text-to-SQL research emerges to bridge this gap by combining natural language's intuitiveness with SQL's computational power (Katsogiannis-Meimarakis and Koutrika, 2023).

Early approaches utilizing end-to-end neural architectures (Zhong et al., 2017; Xu et al., 2017) faced complex syntax interpretation and semantic disambiguation challenges. The advent of LLMs (Touvron et al., 2023; Achiam et al., 2023) has revolutionized this field, introducing two primary methodological streams: In-Context Learning (ICL) and planning-based methods.

ICL approaches leverage few-shot demonstrations to guide LLMs in SQL generation (Gao et al., 2023). While methods like DAIL-SQL enhance performance through adaptive example selection, their effectiveness remains dependent on demonstration quality and selection strategy. Planning-based methods, evolving from Chain-of-Thought (CoT) prompting (Wei et al., 2022), decompose complex queries into manageable sub-tasks. This approach, exemplified by DINSQL (Pourreza and Rafiei, 2024) and C3SQL (Dong et al., 2023), has demonstrated promising results. Recent error correction mechanisms (Shinn et al., 2024; Yao et al., 2023) further enhance robustness through environment feedback and task re-planning.

While these approaches have advanced the field, they face three key limitations: error propagation due to tightly coupled sub-tasks, computational overhead from iterative correction processes, and suboptimal decomposition strategies resulting from linear execution patterns. Our work addresses these challenges through a hybrid framework combining tree-structured planning with reinforcement learning optimization, enabling flexible error recovery while maintaining computational efficiency.

3 Methodology

Complex text-to-SQL tasks face three main challenges: query decomposition, error recovery, and schema understanding. Chain-based methods often fail due to linear execution and limited error resilience, while optimizing closed-source LLMs for specific tasks incurs high operational costs.

To address this, we propose TALENT, a hybrid framework integrating: (1) A tree-structured task planning integrated with a knowledge graph (KG)-based schema linking mechanism that enables flexible task decomposition, adaptive error recovery, and accurate database understanding; (2) A reinforcement learning module that refines decomposition strategies based on historical patterns. The system's architecture, illustrated in Fig. 1, demonstrates how these components collaboratively achieve both execution accuracy and computational efficiency.

3.1 Tree-based Task Planning

Our tree-structured task planning system consists of three key components: (1) a hierarchical task decomposition mechanism, (2) a dynamic error recovery module, and (3) a knowledge graph-driven schema linking approach. These components work collaboratively to break down complex queries, handle execution failures, and ensure accurate schema understanding.

3.1.1 Task Tree Construction

While chain-based mechanisms like Chain of Thought (CoT) (Wei et al., 2022) decompose tasks linearly, complex query processing requires a more comprehensive approach (Khot et al., 2022). We propose a hierarchical decomposition framework that progressively breaks tasks into manageable sub-tasks. The construction process consists of three main phases, as illustrated in Fig. 2:

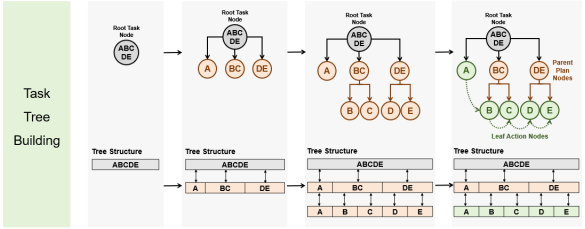

Figure 2: Detail Process of Task Tree Building

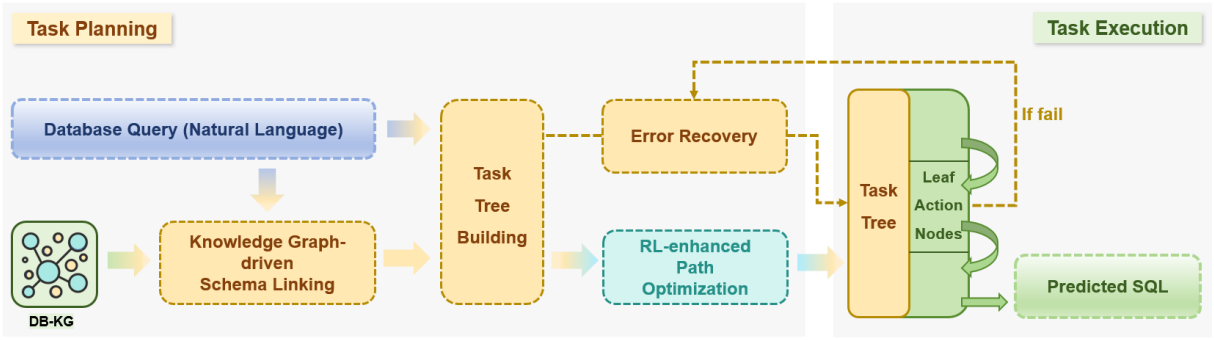

Figure 1: General architecture of TALENT

Initial Tree Formation Given a complex task (e.g., "ABCDE"), our system first establishes a root node containing the complete task description. This serves as the starting point for subsequent decomposition.

Recursive Decomposition For each non-leaf node, we perform:

1) **Complexity Assessment:** The system evaluates task complexity through:

$$C = \underset{C \in \{0,1\}}{\text{argmax}}\ P(C|x, R), \tag{1}$$

where x denotes the task description, C indicates complexity (1 for complex, 0 for simple), and R represents predefined evaluation criteria. The assessment considers structural complexity regarding required tables and joins, semantic complexity regarding nested queries and aggregations, and data manipulation complexity based on operation types.

2) **Sub-task Generation:** For tasks identified as complex ($C = 1$), the system generates optimal sub-tasks according to:

$$\begin{aligned} P &= \underset{p \in \mathcal{P}}{\text{argmax}}\ P(p|x, S) \\ &= \prod_{t=1}^{n} P(p_t|x, S, p_1, ..., p_{t-1}), \end{aligned} \tag{2}$$

where x is the query, S is schema information, and $\{p_1, p_2, ..., p_n\}$ denotes the sub-task sequence.

Leaf Node Formation The decomposition process continues recursively until all sub-tasks are classified as "simple". These terminal nodes form the executable components of our task tree.

This tree-structured design achieves dual benefits. The sequential connectivity of leaf nodes naturally generates executable plans. In contrast, parent nodes enable complete traceability across the execution pathway, facilitating efficient task execution while providing robust error recovery capabilities, as discussed in the following section.

3.1.2 Error Recovery Mechanism

Chain-based correction mechanisms like ReACT and Reflexion struggle with structural-level error recovery due to their linear execution nature (Ji et al., 2024). We propose an error recovery mechanism that leverages our tree-structured architecture to enable dynamic error recovery. The process consists of three phases, as illustrated in Fig. 3:

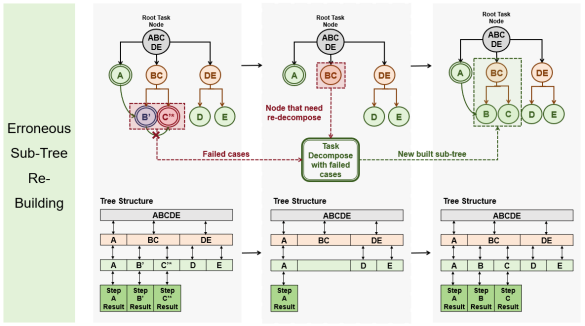

Figure 3: Detail Process of Erroneous Sub-Tree Re-Building

Error Detection and Assessment When node execution fails (e.g., node C'^* in Fig. 3), our system first attempts standard ReACT recovery. If the error persists, the associated sub-tree is marked as "Erroneous," triggering our specialized recovery mechanism.

Selective Tree Reconstruction The recovery proceeds in two steps:

1) **Pruning Operation:** Inspired by CART's pruning methodology (Lewis, 2000), the system performs targeted removal of affected components while preserving valid executions:

$$T_{pruned} = T - S(C'^*) \tag{3}$$

where $S(C'^*)$ represents sibling nodes of the error node, and T denotes the original task tree. This selective pruning ensures minimal disruption to successfully executed branches.

2) **Sub-tree Regeneration:** The system reconstructs the affected sub-tree using structured error information:

$$\begin{aligned} T' &= T_{pruned} \cup T'_{subtree} \\ &= T_{pruned} \cup \mathcal{M}_{rebuild}(C'^*, B', E(C'^*)), \end{aligned} \tag{4}$$

where $\mathcal{M}_{rebuild}$ represents our specialized LLM agent that incorporates error context $E(C'^*)$ to generate optimized recovery plans.

Execution Resumption Following successful sub-tree reconstruction, execution resumes from node B, ensuring seamless integration of the recovered branch with the existing execution state.

This recovery mechanism achieves two key advantages. First, it enables structural-level error handling, significantly enhancing recovery robustness. Second, it maintains computational efficiency by preserving valid execution states and unaffected branches. These capabilities form the foundation for our subsequent schema-linking mechanism, further enhancing query understanding.

3.1.3 Knowledge Graph-driven Schema Linking

Database schema understanding is crucial for SQL generation. However, both insufficient and redundant contextual information can impair the model performance, leading to the "illusion phenomenon" (Ji et al., 2023). To address this, we propose a lightweight, property-oriented knowledge graph module that enhances schema understanding while maintaining computational efficiency.

Unlike conventional KG approaches that require extensive model training (Chen et al., 2020), our knowledge graph requires no model training to fit in dynamic database environments (See Section A.1 for more details), automatically extracts and organizes database components (tables, columns, and relationships) into semantic triples. We employ a two-stage inference process to identify relevant schema information:

$$KG_{opt} = f_{filter}(f_{table}(KG_{full}, q), q), \tag{5}$$

where KG_{full} represents the complete database schema, q is the input query, f_{table} identifies relevant tables, and f_{filter} selectively includes columns when the table count exceeds a predefined threshold τ (empirically set as 2 for Spider dataset).

The effectiveness of our KG-driven approach is particularly evident in disambiguating semantically ambiguous queries. Consider the ambiguous query: "Find students who took Biology and Chemistry." This could mean either (a) students enrolled in both subjects or (b) students enrolled in at least one subject. As shown in Fig. 4, our KG representation naturally resolves this ambiguity through its one-to-many relationship between students and courses, favoring the conjunctive interpretation.

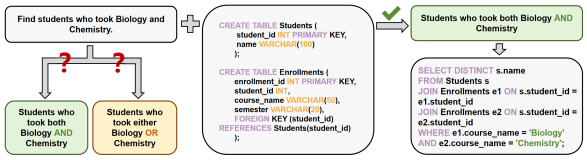

Figure 4: Database Schema's Role in Query Disambiguation

This mechanism ensures that only the most pertinent schema information is provided to the execution agents, effectively addressing information insufficiency and redundancy problems while maintaining LLM compatibility.

3.2 Path Optimization via Reinforcement Learning

While our tree-based framework facilitates task decomposition, optimizing execution paths requires a systematic approach (Latif, 2024). Although LLMs are effective in decomposition and execution, they face limitations in computational efficiency.

We propose a curriculum-driven reinforcement learning framework that progressively optimizes decomposition strategies through historical execution patterns. This approach combines supervised preheating for establishing basic capabilities with curriculum-based RL for adaptive optimization (Silver et al., 2014), ensuring efficiency and adaptability.

3.2.1 Problem Formulation

Our solution employs a transformer-based architecture with a policy network $f_\theta : S \to \Delta(A)$ mapping states to action distributions, and a value network $v_\phi : S \to \mathbb{R}$ estimating expected returns. Given a problem instance $x \in X$ and initial state s_0, we aim to find an optimal policy π^* that maximizes the objective:

$$\pi^* = \arg\max_{\pi \in \Pi} \frac{V_\pi(s_0)}{\log(1 + \lambda C(\pi))} \tag{6}$$

where $C(\pi)$ denotes the expected resource cost, $V_\pi(s_0)$ represents the expected cumulative reward under policy π:

$$V_\pi(s_0) = \mathbb{E}_{\tau \sim \pi} \left[\sum_{t=0}^{T} \gamma^t R(s_t, a_t) | s_0 \right] \tag{7}$$

3.2.2 Training Process

Due to the large discrete action space and the multi-objective nature of our task, direct reinforcement learning in such a setting would be unstable, as random exploration rarely yields successful paths, and the reward signals are inherently sparse (Van Hasselt et al., 2016). We implement a two-stage training framework to address these challenges by combining supervised learning's efficiency with reinforcement learning's adaptability (Silver et al., 2016).

Data Collection and Preparation For each question x, we leverage our tree construction module to generate n diverse candidate paths $\{p_1, p_2, ..., p_n\}$. Each path p_i is executed to obtain feedback (c_i, r_i), where $c_i \in \{0, 1\}$ is the success indicator and $r_i = [tokens, APIcalls, time]$ captures resource usage.

Supervised Preheating The first stage establishes basic path generation capabilities through weighted maximum likelihood estimation (WMLE) (Shen et al., 2015). By assigning higher weights to high-quality paths, we guide the model toward solutions that balance success rates and resource efficiency:

$$\begin{aligned} \mathcal{L}_{\text{WMLE}} &= -\mathbb{E}_{(x,p_i)\sim\mathcal{D}} \sum_{t=1}^{T} w_i \cdot \log \pi_\theta(a_t | s_t), \\ w_i &= \frac{\exp(\eta \cdot (c_i - \gamma \|\mathbf{r}_i\|_2))}{\sum_{j=1}^{n} \exp(\eta \cdot (c_j - \gamma \|\mathbf{r}_j\|_2))}, \end{aligned} \tag{8}$$

where a_t is the t-th node in the path p_i, s_t represents the state including the problem x and previously generated nodes, η scales the weight sensitivity, and γ balances correctness and resource efficiency. The resource norm $\|\mathbf{r}_i\|_2 = \sqrt{w_{token} r_{i1}^2 + w_{\text{API}} r_{i2}^2 + w_{\text{time}} r_{i3}^2}$ combines weighted token, API, and time costs.

Curriculum-Driven Reinforcement Learning The second stage employs curriculum learning (Bengio et al., 2009) to shift optimization focus from efficiency to correctness progressively. Following the Spider dataset's complexity categorization, we partition problems into four difficulty levels and design a dynamic reward function inspired by reward-shaping techniques (Wiewiora et al., 2003):

$$\begin{aligned} R(p_i) &= \alpha(s) \cdot c_i + \beta(s) \cdot (1 - \frac{\|\mathbf{r}_i\|_2}{r_{max}}), \\ \alpha(s) &= \alpha_{init} \cdot (1 - \frac{s-1}{4}),\ \beta(s) = 1 - \alpha(s), \end{aligned} \tag{9}$$

where $s \in \{1, 2, 3, 4\}$ is the curriculum stage. Early stages emphasize efficiency (β-weighted), while later stages emphasize execution success (α-weighted).

Our model is fine-tuned using Proximal Policy Optimization (PPO) (Schulman et al., 2017) with a KL penalty scaled by the coefficient factor β_{KL} (Kakade and Langford, 2002). The training objective combines policy improvement, value estimation, and behavioral regularization:

$$\mathcal{L}_{Total} = \mathcal{L}_{policy} + \mathcal{L}_{value} + \beta_{KL} \cdot \text{KL}(\pi_\theta \| \pi_{\theta_{\text{old}}}) \tag{10}$$

The PPO policy loss uses importance sampling with clipping to ensure conservative updates (Schulman et al., 2017):

$$\mathcal{L}_{policy} = \mathbb{E}\left[\min\left(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}} \hat{A}_t, \text{clip}\left(\frac{\pi_\theta}{\pi_{\theta_{\text{old}}}}, 1-\epsilon, 1+\epsilon \right) \hat{A}_t \right) \right], \tag{11}$$

where $\hat{A}_t$ is the advantage estimate comparing observed returns to value predictions, and ϵ controls the policy update magnitude.

3.3 Module Integration and Interaction

TALENT integrates four core components: task tree construction, error recovery, schema linking, and path optimization into a cohesive system through the following interactions:

- **Tree Construction and Error Recovery:** The hierarchical tree structure provides natural checkpoints, enabling targeted error isolation and recovery while preserving valid computations in unaffected branches.
- **KG-driven schema linking:** The KG-driven schema linking enhances decomposition accuracy by providing precise database context during the planning phase, preventing semantic errors before execution.

- **Path Optimization:** The RL optimizer refines decomposition strategies based on execution feedback and historical patterns.

This integrated design creates a self-improving system where each component benefits the overall pipeline performance.

4 Experimental Evaluation

4.1 Experiment Setup

We implement gpt-3.5 and gpt-4 as the base LLM for Tree-based Task Planning, with the temperature set to 0 to ensure output stability. For the Path Optimization component, we employ Llama-3-8B, balancing computational cost, reasoning ability, and RL training efficiency.

Dataset We conduct experiments on a widely-used text-to-SQL benchmark: Spider (Yu et al., 2018). Spider is a comprehensive cross-domain text-to-SQL dataset with 10,181 instances and 5,693 distinct complex SQL queries, categorized based on complexity levels.

Metrics Given the potential for different SQL queries to convey the same semantic concept, we adopt the official Spider execution accuracy (EX) metric, which compares the execution results of predicted SQL and ground truth SQL queries.

Baselines We compared TALENT against two types of baselines:

- LLM Baselines: Zero-shot and few-shot performance of GPT-3.5-turbo and GPT-4.0 [1].
- In-Context Learning Methods: State-of-the-art approaches, including: DINSQL (chain-based decomposition) (Pourreza and Rafiei, 2024), C3SQL (three-stage framework combining CoT with SQL generation) (Dong et al., 2023), and DAILSQL (structure-aware example selection with domain adaptation) (Gao et al., 2023). [2]

We integrate a one-random-prompt text-to-SQL agent for direct SQL generation for a fair comparison (detailed in Section B.3).

[1] results from DAIL-SQL

[2] Since the LLM-powered Agent approaches have higher difficulty in re-producing, to ensure a fair and transparent comparison, we consider the projects that have their predicted gold file released as high-confidence projects and choose them as our baselines.

4.2 Baseline Comparison

We evaluate TALENT through extensive experiments on the Spider development set, comparing it with both pure in-context learning (ICL) approaches (Table 1) and chain-based methods (Table 2). Our results demonstrate that planning mechanisms significantly enhance performance on complex tasks, and TALENT's tree-structured approach outperforms existing planning strategies across different model architectures and sample settings.

Execution Accuracy (EX) - Pure ICL Approaches			
LLM	**Planning**	**ICL Approach**	**Overall**
GPT-3.5	/	0-shot	74.4
	/	1-shot (Random)	73.9
	/	3-shot (Random)	73.6
	TALENT (Ours)	1-shot (Random)	75.0
	/	5-shot (DAIL-SQL)	75.7
GPT-4.0	/	0-shot	72.3
	/	1-shot (Random)	77.4
	/	3-shot (Random)	79.4
	/	5-shot (DAIL-SQL)	83.1
	TALENT (Ours)	**1-shot (Random)**	**83.2**

Table 1: Spider Execution Rate (EX) - Pure In-Context Learning Approaches

Execution Accuracy (EX) - Chain based ICL Approaches			
LLM	**Planning**	**ICL Approach**	**Overall**
GPT-3.5	TALENT (Ours)	1-shot (Random)	75.0
	CoT	0-shot + C3SQL	81.8
GPT-4.0	CoT	0-shot + DIN-SQL	72.9
	CoT	14-shot + DIN-SQL	82.8
	TALENT (Ours)	**1-shot (Random)**	**83.2**

Table 2: Spider Execution Rate (EX) - Chain based ICL Approaches

Conventional approaches show clear performance limitations in pure ICL scenarios (Table 1). With GPT-3.5, increasing demonstration examples from zero-shot to three-shot yields only marginal improvements in execution accuracy (+0.8%). In contrast, TALENT achieves 75.0% accuracy with single-shot random examples, surpassing the three-shot setting by 1.4%. This improvement is more pronounced with GPT-4.0, where TALENT achieves 83.2% accuracy using one-shot examples, outperforming even the carefully curated five-shot DAIL-SQL approach (83.1%). These results demonstrate how structured planning can overcome the limitations of traditional ICL methods through systematic task decomposition.

The advantages of planning mechanisms become more evident in chain-based scenarios (Table 2). On GPT-4.0, TALENT achieves state-of-the-art accuracy (83.2%) with one-shot examples, surpassing DIN-SQL's performance (82.8%), requiring 14 demonstration examples. This validates TALENT's ability to reduce error propagation through non-linear inference paths, offering superior robustness compared to linear chain reasoning.

TALENT demonstrates three key advantages over existing approaches:

- **Sample Efficiency**: Under identical model conditions, TALENT achieves 83.2% accuracy with single-shot random examples, while DAIL-SQL and DIN-SQL require 5 and 14 examples to achieve comparable performance. This demonstrates TALENT's ability to maximize information utility through dynamic planning strategies.

- **Task Generalization**: Unlike DAIL-SQL and DIN-SQL, which rely on domain-specific optimizations, TALENT achieves comparable performance using random demonstration examples. This demonstrates its effectiveness in establishing a generalizable problem-solving framework through structured planning.

- **Model Compatibility**: TALENT shows consistent improvements across model architectures, with performance gains on GPT-3.5 and GPT-4.0, respectively, indicating its capability to leverage larger model capacities.

These results validate TALENT's effectiveness in combining structured planning with ICL, offering a robust and efficient approach for text-to-SQL tasks. The method's strong performance with minimal demonstrations makes it particularly valuable, as high-quality examples are usually limited.

4.3 Analysis of Spider Development Set Inconsistencies

Following our baseline experiments, a detailed analysis of the evaluation results revealed several inconsistencies in the Spider development set's ground truth answers.

We identified three primary types of inconsistencies through examination: semantic discrepancies, misspelled words, and grammatical errors. Fig. 5 illustrates a representative case.

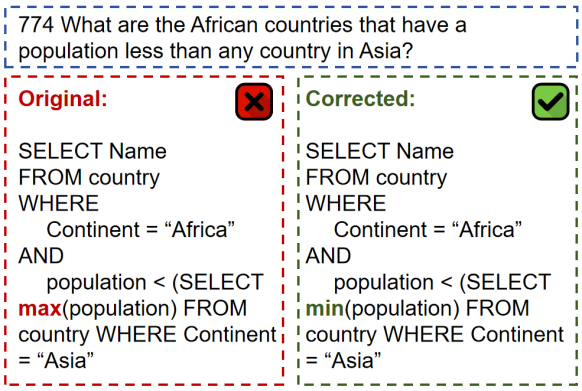

Figure 5: Example of ground truth inconsistency in Spider DEV set: incorrect aggregation operator in a query comparing population statistics between continents

To address these issues, we developed a correction algorithm that rectifies these inconsistencies while preserving the dataset's fundamental evaluation principles. The complete implementation details are available in our code repository.

Performance Analysis We gained insights from applying our corrected evaluation metrics (Table 3). While most approaches show improvements in EASY queries (+0.8% to +1.6% points), the performance variations become more pronounced in complex categories, where TALENT shows consistent improvements in both categories, while other approaches show mixed results.

Methodology Analysis The results suggest that underlying methodologies would strongly influence robustness:

- **Chain-based Fixed Few-shot Methods (DIN-SQL)**: Using few-shot prompts with identical examples for all queries, DIN-SQL shows the most significant performance decrease under corrected metrics, suggesting its sensitivity to dataset-specific biases.

- **Dynamic Few-shot Methods (DAIL-SQL)**: Despite using only 5-shot prompts, DAIL-SQL's dynamic example selection strategy shows robust performance, indicating the advantages of adaptive example selection in generalization improvement.

- **Multi-stage Generation (C3)**: The three-stage SQL generation approach demonstrates stable performance improvements, with notable gains in the MEDIUM and HARD categories. This indicates that the multi-stage

Corrected Spider Execution Rate (EX) - DEV Set															
LLM	**Approach**	**EASY**			**MEDIUM**			**HARD**			**EXTRA**			**OVERALL**		
		Origin	New	Change	Origin	New	Change	Origin	New	Change	Origin	New	Change	Origin	New	Change
CodeX davinci	DIN-SQL	93.1	**94.7**	1.6	81.6	81.2	-0.4	69	70.8	1.8	50.6	47.6	-3	77.3	75	-2.3
GPT-3.5	TALENT (Ours)	91.5	92.3	0.8	84.5	84.9	0.4	56.3	59.3	3	44.6	45.8	1.2	75	76.2	1.2
GPT-4	DIN-SQL	92.3	93.1	0.8	87.4	83.4	-4	76.4	74.6	-1.8	**62.7**	56.1	-6.6	82.8	79.9	-2.9
GPT-3.5	C3	92.7	92.7	0	85	87	2	**77.6**	**79.4**	1.8	62	**63.8**	1.8	81.8	83.3	1.5
GPT-4	DAIL-SQL	91.5	92.3	0.8	89.2	91.2	2	77	78.8	1.8	60.2	60.8	0.6	**83.2**	84.7	1.5
GPT-4	**TALENT (Ours)**	**93.5**	94.3	0.8	**93.3**	**95.7**	2.4	69.5	72.5	3	54.8	60.2	5.4	**83.2**	**85.8**	2.6

Table 3: Comparison of original and corrected execution accuracy (EX) on the Spider development set. Green indicates positive changes in performance under corrected metrics, while red suggests in the opposite.

reasoning process helps maintain robustness in SQL generation tasks.

- **Tree-structured Planning (TALENT)**: Our approach shows the most consistent improvements across all categories, including strong gains in complex queries. The tree-structured decomposition provides better robustness against dataset artifacts, maintaining high performance on corrected metrics.

These findings highlight the importance of choosing a methodology to develop robust text-to-SQL systems. While fixed prompting strategies achieve high performance on standard metrics, more flexible and adaptive approaches appear to have better generalization ability. Our corrected evaluation framework provides a more reliable benchmark for assessing true model capabilities and reveals the advantages of adaptive decomposition strategies in complex text-to-SQL tasks.

4.4 Ablation Studies

Our systematic ablation study evaluates the contribution of individual modules to system performance. The results presented in Table 4 illustrate a significant decrease in overall performance when any of these modules are omitted.

The tree construction framework proves particularly crucial for complex query processing. Its absence forces the model to rely on basic coT planning, significantly weakening its capacity for structured query resolution and self-correction, leading to a substantial performance drop, especially in Hard and Extra-difficulty queries.

Similarly, removing KG-driven schema linking introduces input redundancies and negatively impacts SQL generation, particularly in medium-to-hard queries, where schema understanding is crucial. The Path Optimization module also proves essential, as its removal leads to a consistent performance decline across all difficulty levels, highlighting its role in guiding query execution and ensuring overall system reliability.

Model	Approach	EASY	MEDIUM	HARD	EXTRA	OVERALL
TALENT (GPT-4)	**Default**	**94.3**	**95.7**	**72.5**	**60.2**	**85.8**
	w/o KG	92.3	92.7	66.7	53.0	81.9
	w/o Path Optimization	91.9	87.2	65.6	44.6	77.2
	w/o Error Recovery	92.7	87.2	60.3	44.6	78.0
	w/o Tree Construction	91.9	91.5	58.0	42.2	77.2
TALENT (GPT-3.5)	**Default**	**92.3**	**84.9**	**59.3**	**45.8**	**76.2**
	w/o KG	90.3	82.1	54.6	41.0	72.8
	w/o Path Optimization	88.7	81.6	54.0	40.4	72.0
	w/o Error Recovery	89.9	81.8	51.1	36.7	71.3
	w/o Tree Construction	89.1	82.3	48.3	30.7	69.9

Table 4: Ablation study on TALENT's performance (EX) on the development set, with and without each module.

5 Conclusion & Limitations

This work addresses three fundamental challenges in complex text-to-SQL tasks: query decomposition, error recovery, and schema understanding. Traditional chain-based approaches often struggle with these challenges due to their linear execution patterns and limited error resilience. To overcome these limitations, we present TALENT, a hybrid framework that combines tree-structured planning with knowledge graph-based schema linking and reinforcement learning optimization.

Experiments on the Spider benchmark demonstrate TALENT's effectiveness, and our analysis of Spider development set inconsistencies shows that TALENT's adaptive decomposition strategy and error recovery mechanism outperforms fixed template-based methods. Furthermore, integrating reinforcement learning enhanced path optimization further enhances the system's ability to refine decomposition strategies.

Despite these strengths, scaling to enterprise databases, particularly in maintaining reasoning efficiency with increased schema complexity, remains a challenge. Future work will focus on enhancing the scalability of our method and exploring more sophisticated reinforcement learning algorithms for decomposition optimization.

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*.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pages 41–48.

Zhe Chen, Yuehan Wang, Bin Zhao, Jing Cheng, Xin Zhao, and Zongtao Duan. 2020. Knowledge graph completion: A review. *Ieee Access*, 8:192435–192456.

Edgar F Codd. 1974. *Seven steps to rendezvous with the casual user*, volume 1333. IBM Corporation.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, et al. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-sql empowered by large language models: A benchmark evaluation. *arXiv preprint arXiv:2308.15363*.

Zhenlan Ji, Daoyuan Wu, Pingchuan Ma, Zongjie Li, and Shuai Wang. 2024. Testing and understanding erroneous planning in llm agents through synthesized user inputs. *arXiv preprint arXiv:2404.17833*.

Ziwei Ji, Tiezheng Yu, Yan Xu, Nayeon Lee, Etsuko Ishii, and Pascale Fung. 2023. Towards mitigating llm hallucination via self reflection. In *Findings of the Association for Computational Linguistics: EMNLP 2023*, pages 1827–1843.

Sham Kakade and John Langford. 2002. Approximately optimal approximate reinforcement learning. In *Proceedings of the Nineteenth International Conference on Machine Learning*, pages 267–274.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*, 32(4):905–936.

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406*.

Ehsan Latif. 2024. 3p-llm: Probabilistic path planning using large language model for autonomous robot navigation. *arXiv preprint arXiv:2403.18778*.

Roger J Lewis. 2000. An introduction to classification and regression tree (cart) analysis. In *Annual meeting of the society for academic emergency medicine in San Francisco, California*, volume 14. Citeseer.

Pengfei Liu, Weizhe Yuan, Jinlan Fu, Zhengbao Jiang, Hiroaki Hayashi, and Graham Neubig. 2023. Pre-train, prompt, and predict: A systematic survey of prompting methods in natural language processing. *ACM Computing Surveys*, 55(9):1–35.

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

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. 2017. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*.

Shiqi Shen, Yong Cheng, Zhongjun He, Wei He, Hua Wu, Maosong Sun, and Yang Liu. 2015. Minimum risk training for neural machine translation. *arXiv preprint arXiv:1512.02433*.

Tianze Shi, Kedar Tatwawadi, Kaushik Chakrabarti, Yi Mao, Oleksandr Polozov, and Weizhu Chen. 2018. Incsql: Training incremental text-to-sql parsers with non-deterministic oracles. *arXiv preprint arXiv:1809.05054*.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2024. Reflexion: Language agents with verbal reinforcement learning. *Advances in Neural Information Processing Systems*, 36.

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, Guy Lever, Nicolas Heess, Thomas Degris, Daan Wierstra, and Martin Riedmiller. 2014. Deterministic policy gradient algorithms. In *International conference on machine learning*, pages 387–395. Pmlr.

Mirac Suzgun, Nathan Scales, Nathanael Schärli, Sebastian Gehrmann, Yi Tay, Hyung Won Chung, Aakanksha Chowdhery, Quoc V Le, Ed H Chi, Denny Zhou, et al. 2022. Challenging big-bench tasks and whether chain-of-thought can solve them. *arXiv preprint arXiv:2210.09261*.

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*.

Hado Van Hasselt, Arthur Guez, and David Silver. 2016. Deep reinforcement learning with double q-learning. In *Proceedings of the AAAI conference on artificial intelligence*, volume 30.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, 35:24824–24837.

Eric Wiewiora, Garrison W Cottrell, and Charles Elkan. 2003. Principled methods for advising reinforcement learning agents. In *Proceedings of the 20th international conference on machine learning (ICML-03)*, pages 792–799.

Xiaojun Xu, Chang Liu, and Dawn Song. 2017. Sqlnet: Generating structured queries from natural language without reinforcement learning. *arXiv preprint arXiv:1711.04436*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2023. React: Synergizing reasoning and acting in language models. In *International Conference on Learning Representations (ICLR)*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics.

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

A Extensive Experiments

A.1 Cross-domain Evaluation on Economic Database

A.1.1 Experimental Setup

To evaluate TALENT's cross-domain generalization capability, we conducted extensive experiments on a real-world macroeconomic database. This database consists of 46 interconnected tables containing various economic indicators, including GDP metrics, financial indices, and commodity prices across multiple countries.

The evaluation followed the Spider benchmark's difficulty stratification, with: 10 easy queries (executed 3 times each), 10 medium queries (executed 3 times each), 3 hard queries (executed 10 times each), 3 extra hard queries (executed 10 times each).

We evaluated our approach against three established LLM-driven agent planning methodologies: Chain-of-thought (Wei et al., 2022), ReACT (Yao et al., 2023), and Reflection (Shinn et al., 2024). The evaluation was conducted in two phases: Initially, these methodologies were implemented without our schema linking module, utilizing standard prompt templates with complete database schema information as input. Subsequently, we integrated our KG-driven Schema Linking methods into these approaches to assess performance changes.

A.1.2 Dataset Characteristics

The economic database exhibits three key characteristics that distinguish it from the Spider dataset:

- Scale Complexity: Contains extensive time-series data with multiple interconnected economic metrics
- Naming Diversity: Employs domain-specific terminology and varied naming conventions
- Schema Complexity: Features complex relationships between different economic indicators

A.1.3 Results and Analysis

The experimental results in Table 5 demonstrate TALENT's promising performance in handling complex real-world databases. Two key observations emerge:

- The KG Schema Linking module consistently improves performance across all approaches, with particularly significant gains in medium and hard queries.
- TALENT maintains its performance advantage across all difficulty levels, showing particular strength in complex queries where baseline methods struggle significantly.

A.2 Ablation Study: RL vs. LLM Agent in Path Optimization

A.2.1 Implementation Details

To validate the effectiveness of our reinforcement learning-based path optimization, we implemented a pure LLM agent-based alternative for comparison. The agent-based approach follows a similar workflow but relies entirely on LLM for path optimization decisions, as illustrated in Figure 6.

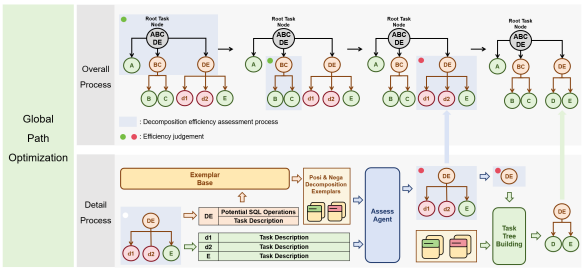

Figure 6: Process of Path Optimization

As illustrated in Fig. 6, the LLM agent-based optimization module operates in three primary stages.

First, it extract each parent node's task description and potential SQL operations (derived from the initial task decomposition phase).

Second, the system would search the exemplar base to identify matching exemplars, including the negative and positive ones.

Third, the process proceeds by sending three key components to our Assess Agent: (1) the selected exemplars, (2) the parent node's task description, and (3) its corresponding sub-tasks.

Through comparative analysis of these components, the Agent evaluates the effectiveness of the current decomposition strategy. When the evaluation indicates suboptimal decomposition, the system triggers a re-decomposition of the parent node. This algorithm is also formally presented in Alg. 1.

A.2.2 Performance Comparison

Comparison between RL and Agent approach			
	Path Optimization Approach	Overall EX	Ave Time Cost
TALENT	LLM Agent	81.9	29.3s
TALENT	Reinforcement Learning	83.2	20.7s

Table 6: Comparison between Reinforcement Learning enhanced and Agent approach

Performance Comparison on Economic Database with and without KG Schema Linking									
LLM	**Approach**	**EASY**		**MEDIUM**		**HARD**		**EXTRA**	
		W/O	**W**	**W/O**	**W**	**W/O**	**W**	**W/O**	**W**
GPT-4	Chain-of-Thought (Wei et al., 2022)	53.3	**70.0**	33.3	**63.3**	0.0	0.0	0.0	0.0
GPT-4	ReACT (Yao et al., 2023)	56.7	**76.7**	46.7	**70.0**	0.0	**11.1**	0.0	0.0
GPT-4	Reflexion (Shinn et al., 2024)	66.6	**86.7**	53.3	**73.3**	11.1	**22.2**	0.0	**11.1**
GPT-4	**TALENT (Ours)**	**86.7**	**93.3**	**80.0**	**90.0**	**66.7**	**77.8**	**46.7**	**55.6**

Table 5: Performances on real-world database environment - with & without KG Schema Linking

The comparative analysis reveals two key advantages of the Reinforcement Learning-based approach:

- **Performance Advantage**: The RL-based method achieves higher execution accuracy (83.2% vs 81.9%), demonstrating more reliable path optimization.
- **Efficiency Gain**: A significant 29.3% reduction in average processing time (from 29.3s to 20.7s) indicates superior computational efficiency.

These improvements can be attributed to: (1) RL model's ability to learn optimal decomposition patterns from historical executions. (2) Reduced token consumption compared to LLM-based agents. (3) More systematic exploration of the solution space.

Algorithm 1 Tree Optimization

Input: tree: treelib.Tree, db_info: dict
Output: tree: treelib.Tree

function IF_REBUILD($parent, tree$)
    pos_case, neg_case = Get_Cases (parent)
    is_correct = Contrastive_Learning (pos_case, neg_case, parent, children)
    return is_correct
end function

function TREE_OPTIMIZATION($tree, db\_info$)
    for parent in tree.parents **do**
        flag = IF_REBUILD (parent, tree)
        if ! flag **then**
            tree.remove_children(parent)
            tree = NODE_GORWING (parent, db_info, tree)
        end if
    end for
    return tree
end function

A.3 Supplementary Materials

A.3.1 Database Schema Details

Sample Database Schema

```
CREATE TABLE ‘101_gdpingermany_namegdp‘ ( ‘time‘ datetime DEFAULT NULL, ‘china‘ double DEFAULT NULL, ‘usa‘ double DEFAULT NULL, ‘japan‘ double DEFAULT NULL, ‘germany‘ double DEFAULT NULL, ‘u.k.‘ double DEFAULT NULL, ‘switzerland‘ double DEFAULT NULL ) ENGINE = InnoDB DEFAULT CHARSET = utf8mb4 COLLATE = utf8mb4_general_ci;
```

A.3.2 Knowledge Graph Structure

Knowledge Graph Sample

```
[time, data_type, datetime]
[spot_uaedubai_crudeoil, data_type, double]
[spot_wti, data_type, double]
[spot_brent, data_type, double]
[spot_china_crudeoil, data_type, double]
[spot_opec_crudeoil, data_type, double]
```

A.3.3 Evaluation Tasks

Tasks were carefully designed to cover various complexity levels and economic scenarios. Example tasks include:

Sample Tasks

```
(1) Easy: “I want to know the highest spot price of gold in Japan.”
```

(2) Easy: “I want to know Japan’s actual GDP after year 2000.”
(3) Easy: “Can you show me Japan’s GDP per capita in 2012?”
(4) Easy: “How about the unemployment rate of Japan after year 2010?”
(5) Easy: “I would like to know how much foreign direct investment the U.K. had in 2008.”
(6) Easy: “How much foreign exchange storage does the U.S.A have in 2020?”
(7) Easy: “What was Japan’s stock market index in 1950?”
(8) Easy: “What is the lowest energy price in Switzerland?”
(9) Easy: “I’d like to see the five highest spot cotton prices in Japan.”
(10) Easy: “In how many years has Japan’s GDP per capita above 10,000?”
(11) Medium: “I’d like to see the years when Japan’s actual GDP was higher than its name GDP.”
(12) Medium: “Between 1990 and 2020, in how many years did Japan’s GDP per capita exceed 10,000?”
(13) Medium: “For how many times was Japan’s stock market index higher than that of the U.S.A?”
(14) Medium: “How much lower is the highest actual GDP in Japan than the highest actual GDP in the US?”
(15) Medium: “What are Germany’s stock market indexes when its industrial production index is above 10?”
(16) Medium: “When Japan’s real GDP is greater than 10, what are the interest rate for the corresponding times?”
(17) Medium: “What are the corresponding energy prices of Japan when its savings rate is lower than 45?”
(18) Medium: “What was Japan’s highest spot gold price during Japan’s cotton futures prices were above 13000?”
(19) Medium: “What is the highest GDP per capita in Germany in years when the Japanese stock market index is above 30,000?”
(20) Medium: “What is the average GDP per capita in Japan in years when the Japan’s house price index is higher than 25?”
(21) Hard: “I want to know the Japan’s stock market index at the time Japan’s house price index reached its highest pitch.”
(22) Hard: “What were Japan’s interest rate and consumer price index when Japan’s unemployment rate peaked?”
(23) Hard: “What is the average spot cotton price of Japan in years when the Japan’s house price index is above 25 and the U.K. frequent account balance is above 120?”
(24) Extra: “I want to see the spot London prices of all metals at 2012-12-12.”
(25) Extra: “I’d like to see Japan’s spot prices of grains(wheat, corn, soybeans) on July 1, 2022.”
(26) Extra: “I want to know the historical highest spot London prices of all metals.”

B Prompt Design

B.1 Tree Construction

Prompt 1: Task Decompose

Now you are a professional SQL engineer, I’m going to give you a [original task] and a [database information] to accomplish the task. If you think the [original task] is too hard for one LLM to finish, then try to decompose the [original task] into several [sub-tasks]. Your one action should follow the framework below:
input:
[original task]: data selection task, given externally.

[database information]: Database information, given externally
output:
[thinking]: Fully based on the given [original task], please think step by step, find out how to decompose the [original task] reasonably.
[Decompose or not]: Answer 'yes', or 'no'
[Sub-tasks]: Sub-tasks that are decomposed from the [original task]. If the task don't need to be decomposed, leave this part empty.

B.2 Knowledge Graph

Prompt 2: Related KG Searching

I'm going to give you a goal and several tables, you need to think step by step, and help me find the tables that may be used when accomplishing the goal. As you only know the table name, to make sure all needed tables are included, you should think more carefully and return all the relevant tables. for each [goal] provided below, extract and list the tables that might be used.
Input:
[goal]: The goal that should be accomplished, given externally.
[tables]: All the data tables that are provided in the database, given externally.
Output:
[thinking]: Your thinking about related tables.
[related tables]: The list of related tables.

B.3 Task Execution Agents

Prompt 3.1: Code Checking

You are a professional SQL engineer, I'm going to give you a task, a sql code and a description of database. According to the given information, tell me if the code could finish the task, if not, try to fix it, or just tell me "correct". Your one action should follow the framework below:
input:
[task]: Task that should be finished, given externally.
[SQL code]: Half-completed SQL code, given externally.
[database description]: Description of database, given externally.
output:
[thinking]: According to the given [task], [SQL code], and [database description], think step by step, figure out if the code given to you could achieve the task, if not, where is the problem.
[correct or not]: Based on the given information and your thinking, if the code is correct, return yes, if not, return no.(Just return "yes" or "no"!)
[final SQL]: SQL code that you generated.

Prompt 3.2: Bug Fixing

You are a professional SQL engineer, I'm going to give you a SQL code, a description of database, and a error message. According to the given information, modify the SQL code. Your one action should follow the framework below:
input:
[SQL code]: Half-completed SQL code, given externally
[database description]: Description of database, given externally
[error message]: Error caused when using the SQL code
output:
[thinking]:According to the given [sql code], [database description] and [error message], think step by step, figure out where is the problem and give a suggestion on code modification.

[final SQL]: SQL code that you generated.

Prompt 3.3: Code Generating

You are a professional SQL engineer, I'm going to give you a task, a half-completed SQL code and a description of database. According to the given information, generate a SQL code to accomplish the task. Your one action should follow the framework below:
input:
[task]: Task that should be finished, given externally.
[half-completed SQL code]: Half-completed SQL code, given externally.
[tasks already finished]: Tasks that have finished, the [half-completed SQL code] is the code that could accomplish those tasks.
[database description]: Description of database, given externally.
output:
[thinking]: According to the given [task], [half-completed SQL code], and [database description], think step by step, and try to generate a SQL code that could accomplish the task.
[final SQL]: SQL code that you generated.