FinSQL: A Full-Stack Solution for Enterprise-Grade Text-2-SQL Tasks in Financial Scenarios

Anonymous ACL submission

Abstract

While Large Language Models (LLMs) have demonstrated remarkable proficiency in general Text-to-SQL tasks, their application in enterprise-level financial scenarios remains bottlenecked by intricate database schemas, rigorous temporal constraints, and the requirement for deep domain expertise. In this paper, we propose a comprehensive framework tailored for industrial financial analytics. First, we introduce an **enterprise-centric data synthesis pipeline** that mirrors real-world complexity through hybrid schema construction and temporal-consistent data population. Second, we develop an **Agentic SQL Pipeline** that replaces traditional end-to-end generation with a modular architecture comprising a *Data Source* locator, a specialized *SQL Tool*, and an iterative *Error Correction* loop. Finally, we implement a **two-stage multi-agent training paradigm**: a general end-to-end SFT followed by a module-specific SFT and Reinforcement Learning phase using the **GSPO** algorithm. Our empirical results show that our model achieves **89.72%** execution accuracy on a specialized financial benchmark and establishes a new state-of-the-art on the BIRD dev set (49.74%), demonstrating superior generalization and robustness in bridging the gap between academic research and industrial application.

1 Introduction

The automation of financial data analysis via Text-to-SQL has become a critical requirement for modern enterprises seeking to derive actionable insights from massive, multi-dimensional data warehouses. Despite the rapid advancement of large language models (LLMs), existing benchmarks like Spider and BIRD often fail to capture the "messy" reality of corporate financial environments, which are characterized by hundreds of interconnected tables, diverse statistical calibers (e.g., stock vs. flow data), complex temporal dependencies, and domain-specific accounting logic. These intricacies render conventional end-to-end approaches ineffective, as they struggle to maintain holistic schema comprehension and fail to incorporate implicit business constraints.

Recent advances in prompt engineering and multi-turn interaction frameworks have pushed the boundaries of Text-to-SQL performance. Techniques such as schema linking, chain-of-thought reasoning, and self-correction mechanisms have demonstrated measurable improvements on academic benchmarks. Yet, these methods often assume a level of schema simplicity and query clarity that does not hold in practice. When applied to financial scenarios, they exhibit three persistent weaknesses: (i) **incomplete schema grounding**, where models fail to map natural language mentions to the correct combination of fact tables and auxiliary enumeration tables; (ii) **temporal reasoning failures**, particularly in handling cumulative versus periodic data transformations; and (iii) **neglect of implicit constraints**, such as defaulting to headquarters-level aggregates when no branch is specified.

To address these limitations, we introduce **FinSQL**, a full-stack solution tailored for enterprise-grade Text-to-SQL tasks in financial domains. Our approach is grounded in three core innovations that collectively bridge the gap between academic research and industrial deployment.

First, we design an **enterprise-centric data synthesis pipeline** that generates high-fidelity training trajectories mirroring real-world complexity. Unlike prior "SQL-first" methods that prioritize query diversity, our pipeline emphasizes *Hybrid Schema Construction*—combining de novo synthesis with empirically crawled schemas—and *Temporal-Grounded Population* over continuous 3–5 year horizons. This yields 17k verifiable (question, SQL, chain-of-thought) triples that incorpo-

rate realistic data noise, hierarchical branch structures, and multi-frequency time-series data. The resulting schemas average 23.2 tables with up to 31 columns per table, substantially exceeding the complexity of existing benchmarks.

Second, we propose an **Agentic SQL Pipeline** that moves beyond monolithic generation toward a modular "tool-use" philosophy. Rather than relying on brittle schema linking, we introduce a dedicated *Data Source* module that first identifies relevant table subsets before SQL generation. This macro-level awareness ensures the model operates within a high-signal context, reducing hallucinations caused by sparsely documented dimensions. Complemented by a specialized *SQL Tool* and lightweight *Error Correction* loop, our pipeline achieves 92.55% accuracy on internal benchmarks, with ablation studies confirming that the *Data Source* module contributes a 12.76% improvement over granular schema linking alone.

Third, we introduce a **two-stage multi-agent training paradigm** that systematically equips the model with both general analytical capabilities and domain-specific execution skills. Stage 1 employs end-to-end supervised fine-tuning on 32k interaction trajectories to establish foundational competencies in problem decomposition and format adherence. Stage 2 targets the SQL module specifically, combining knowledge-augmented SFT on 17k complex pairs with reinforcement learning via **GSPO (Group Sequence Policy Optimization)** on 4k rejection-sampled trajectories. This targeted optimization addresses the "last mile" of execution correctness, particularly in handling cumulative calculations, enumerated value disambiguation, and implicit business constraints.

Our empirical results demonstrate the efficacy of this holistic approach. On our specialized financial benchmark, FinSQL achieves **89.72%** execution accuracy, substantially outperforming models trained on generic datasets such as Spider and OmniSQL. Crucially, this domain-specific tuning does not compromise generalization: on the BIRD dev set, FinSQL achieves **49.74%**, demonstrating robust performance on out-of-domain benchmarks. Error analyses reveal that our two-stage training progressively resolves the three persistent weaknesses identified earlier, with RL optimization proving particularly effective for complex financial logic.

2 Related Works

To enhance the capability of models in Text-to-SQL tasks, existing work has explored various avenues, including **prompt engineering** (e.g., refining input database information, post-processing of queries, and incorporating chain-of-thought reasoning), **multi-turn interaction pipelines** (e.g., self-querying chain-of-thought, error correction, and consistency voting) and so on.

Prompt Engineering 1: Database Schema Information Concatenation. Text-to-SQL systems fundamentally rely on database schema information to generate accurate SQL queries. Existing approaches differ in how they acquire, represent, and select schema elements. For schema acquisition, some methods perform **offline schema augmentation** by automatically extracting metadata from database catalogs (Shkapenyuk et al., 2025), enriching the schema with foreign key relationships, data types, and value distributions. In contrast, recent approaches like AutoLink (Wang et al., 2025c) and CHASE-SQL (Pourreza et al., 2024) adopt **online schema exploration**, where agents interactively probe the database through multiple query rounds to discover schema components relevant to the user's question. Regarding **input format**, most systems follow the DDL (Data Definition Language) convention used in BIRD (Li et al., 2023) and Agentar-Scale-SQL (Wang et al., 2025b), while XiYan-SQL (Liu et al., 2026) introduces M-Schema, a more structured schema representation that explicitly captures table relationships. For **schema selection**, three primary strategies emerge: (1) **schema linking** approaches such as C3-SQL (Dong et al., 2023), MCS-SQL (Lee et al., 2025), and XiYan-SQL (Liu et al., 2026) retrieve only question-relevant tables and columns, though Maamari et al. argue that with sufficiently powerful LLMs, schema linking may become unnecessary when the full schema fits in context; (2) **adaptive schema filtering** methods like DSR-SQL (Hao et al., 2025) iteratively refine schema selection through multiple model calls; and (3) **graph-based retrieval** as in GRAST-SQL(Dat Hoang et al., 2025) employs Graph Transformers for similarity scoring and Steiner tree solutions to identify optimal schema subsets.

Prompt Engineering 2: Query Post-Processing. Query processing encompasses question understanding and retrieval augmentation strategies. **Question classification** approaches,

exemplified by DIN-SQL (Pourreza and Rafiei, 2023), design distinct prompting strategies for simple versus nested/complex questions. **Question structure extraction** methods parse questions to extract key components following predefined patterns, as demonstrated in DIN-SQL and Agentar-Scale-SQL (Wang et al., 2025b). **Example retrieval** has evolved from static few-shot demonstrations to dynamic retrieval mechanisms: CHASE-SQL (Pourreza et al., 2024) constructs online example pools, while XiYan-SQL (Liu et al., 2026) and MCS-SQL (Lee et al., 2025) employ masked question similarity for example recall, significantly improving generalization to unseen questions.

Prompt Engineering 3: Chain-of-Thought Organization. The organization of reasoning chains critically impacts SQL generation quality. The predominant approach follows **step-by-step reasoning**, where models sequentially decompose the generation task (most of the papers). Recent work introduces **structured chain-of-thought** formats that enforce systematic reasoning patterns (Thaker and Bresler, 2025). **Question decomposition** strategies, employed by MAC-SQL (Wang et al., 2025a) and CHASE-SQL (Pourreza et al., 2024), first break complex questions into sub-problems, generate sub-queries, and then synthesize them into final SQL solutions, effectively managing multi-step reasoning requirements.

Multi-Turn Interaction Organization. Multi-turn frameworks address the iterative nature of complex query generation. **Query plan chain-of-thought** approaches, implemented in CHASE-SQL (Pourreza et al., 2024) and AutoLink (Wang et al., 2025c), progressively explore database schemas by simulating execution plans. **Error correction** mechanisms represent a common practice where systems detect and refine syntax errors or empty results through iterative refinement. **SQL consistency voting** strategies, as in CSC-SQL (Sheng and Shuai, 2025) and ReFoRCE (Deng et al., 2025), generate multiple candidates and select through majority vote or consensus enforcement; when disagreements persist, these systems incorporate query plan reasoning to generate refined alternatives. Finally, **scoring model** approaches, exemplified by Agentar-Scale-SQL (Wang et al., 2025b), employ specialized models to pairwise compare candidates based on database information, questions, SQL statements, and execution results, selecting the highest-scoring query. The integration of process-supervised rewards (Zhang et al., 2025) and reinforcement learning with partial rewards (Pourreza et al.) has further enhanced selection quality in recent frameworks.

3 Method

3.1 Evaluation Benchmark

To better characterize financial Text-to-SQL tasks, we construct an **enterprise-oriented benchmark** through database synthesis and real-world query collection. This benchmark simulates the full analytical workflow: problem decomposition, data retrieval, code implementation, and report generation. Evaluation is performed via LLM-as-a-judge against golden answers.

Database Synthesis: We simulate banking activities (deposits, loans) with aggregations at customer, product, and branch levels, and enterprise operations (sales, costs) with corresponding financial statements. Synthesis incorporates real-world complexities: (i) **hierarchical structures**, (ii) **enumeration issues** (e.g., IDs vs. names across tables), and (iii) **data noise** (nulls, outliers, inconsistent calibers).

Tasks: The benchmark includes diverse SQL queries—single/multi-table, sub-queries, financial indicator calculations, and attribution analysis—with **ambiguous expressions and implicit constraints** reflecting authentic business scenarios.

LLM-as-a-judge challenges and mitigations: (i) **Domain expertise**: Supplemented with clearly defined financial knowledge. (ii) **Schema/query ambiguity**: Resolved by incorporating default constraints into domain knowledge. (iii) **Inconsistent evaluation**: Minimized via **minimal golden answers** to reduce scoring errors. (See Appendix A)

3.2 Text-to-SQL Data Synthesis

3.2.1 Innovations

The proposed data synthesis pipeline is designed to bridge the gap between existing Text-to-SQL benchmarks and complex, enterprise-level financial scenarios.

Our pipeline is guided by three primary innovations that distinguish it from prior works:

Enterprise-level Schema Complexity. Unlike simplified benchmarks, we introduce **auxiliary mapping tables** (mapping IDs to entities like institutions and products) and **fact tables** (containing multi-frequency time-series data). We also incorporate diverse **statistical calibers** (e.g., stock vs. flow,

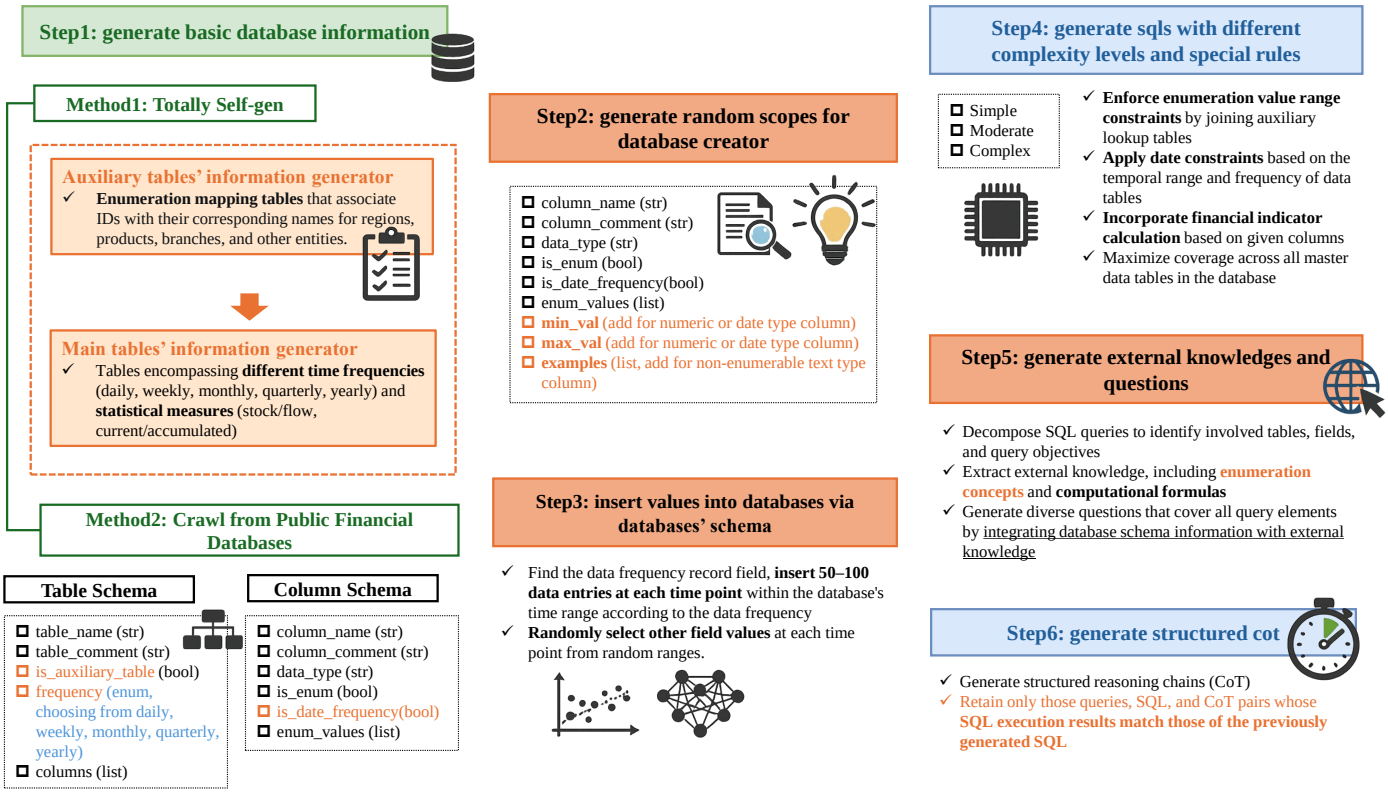


Figure 1: Overview of the four-phase data synthesis pipeline. The pipeline begins with hybrid schema construction (Phase 1), followed by temporal-consistent data population (Phase 2), complexity-stratified SQL generation (Phase 3), and concludes with bidirectional validation and CoT generation (Phase 4).

cumulative vs. period-end), mirroring the intricacy of real-world financial databases.

Table 1 presents the descriptive statistics of our synthesized database schemas, quantifying their complexity. On average, each schema contains 23.2 tables, comprising 7.9 auxiliary tables and 15.3 main tables. The main tables feature an average of 10.2 columns, with some schemas containing up to 31 columns in a single table. This level of schema complexity substantially exceeds that of existing benchmarks (etc. Yu et al., 2018; Li et al., 2023), demonstrating our pipeline's capacity to model enterprise-scale financial databases.

Temporal-Grounded Large-Scale Population. To support execution-based evaluation and trajectory learning, we populate databases over an **extended continuous timeframe**. This ensures that synthesized SQL queries are executed against a more realistic data volume with rigorous temporal alignment.

Diverse SQL, Queries, and External Knowledge Tailored to Real-World Applications. According to OmniSQL (Li et al., 2025), we adopt a "SQL-first" strategy to generate diverse natural language queries based on varied SQL structures. Uniquely, we incorporate **complex temporal SQL syntax, financial indicator calculation** and enforce **temporal and entity constraints**. Furthermore, we generate computational formulas and concepts derived from SQL structures.

3.2.2 Synthesis Methodology

Guided by the aforementioned principles, we implement a four-phase "Schema-First, Execution-Verified" pipeline (Figure 1).

Phase 1: Hybrid Schema Construction We construct diverse schemas through two complementary strategies: (i) **De novo Synthesis** using LLMs to create auxiliary and fact tables, and (ii) **Empirical Crawling** from public financial databases. Each schema includes rigorous metadata: table comments, auxiliary indicators, **data frequencies** (daily to yearly), and **statistical measures** to support complex analytical queries.

Phase 2: Temporal-Consistent Data Population Schemas are populated with synthetic records over a continuous 3–5 year horizon. For each time point, we insert 50–100 entries with values sampled

Table 1: Descriptive Statistics of Dataset Schema Complexity

Statistic	# Aux. Tables	# Main Tables	# All Tables	# Cols. (Main)	Max Cols.	Min Cols.
Mean	7.9	15.3	23.2	10.2	18.0	4.9
Median	7.0	16.0	23.0	10.1	18.0	5.0
Mode	7.0	16.0	23.0	13.9	21.0	4.0
Minimum	0.0	6.0	6.0	5.7	7.0	2.0
Maximum	21.0	31.0	52.0	15.3	31.0	9.0

from predefined ranges, creating a reliable "ground truth" environment for execution verification.

Phase 3: Complexity-Stratified SQL Generation Following OmniSQL (Li et al., 2025), we synthesize SQL-Question pairs across Simple, Moderate, and Complex tiers. Generation enforces four constraints: (i) **Entity** (fact-auxiliary joins), (ii) **Temporal** (time filters), (iii) **Indicator Calculations** (financial metrics), and (iv) **Syntax Diversity** (advanced SQL patterns). We further integrate **enumeration concepts** and **computational formulas** extracted from SQL structures into natural language questions.

Phase 4: Bidirectional Validation and CoT Generation We generate Chain-of-Thought (CoT) sequences outlining the reasoning path, then apply an **Execution-Consistency Filter**: only (Question, SQL, CoT) triples whose execution results match ground truth are retained, ensuring high-fidelity training trajectories.

3.3 End-to-end Strategy Pipline

We propose a modular agentic pipeline (Figure 2) to address financial task complexities, moving beyond standard end-to-end NL2SQL which exhibits persistent weaknesses in interpreting nested schemas, applying multi-level constraints, and utilizing domain knowledge.

Holistic Data Source vs. Granular Schema Linking Our key innovation is the *Data Source* module. While schema linking suffices for academic benchmarks like Spider, industrial contexts with hundreds of sparsely documented columns require holistic table comprehension. Table 2 confirms this: replacing *Data Source* with schema linking drops accuracy from 92.55% to 79.79%.

Implementation Constraints We eschew DDL in prompts due to excessive noise, instead using *Data Source* to focus on 3–4 relevant tables. For latency-sensitive financial analysis, we prioritize single-pass generation quality with a lightweight *Error Correction* loop, avoiding costly self-consistency voting.

Ablation results validate our design: removing *Sqltool* causes the largest drop (to 68.09%), while *Error Correction* provides a vital buffer, pushing accuracy to 92.55%.

Table 2: Ablation study of *Fin-SQL* on our internal financial benchmark. Results represent Execution Accuracy (Ex Acc.) averaged over three independent runs (Avg@3).

Configuration	Bank	Others	Total
Fin-SQL (Full)	**92.13**	**93.94**	**92.55**
w/o *Sqltool*	74.07	48.48	68.09
w/o *Data Source*	87.50	80.30	85.82
w/o *Data Source* (+Schema Link)	84.72	63.64	79.79
w/o *Error Correction*	91.67	85.35	90.19

3.4 Two-stage Training

We adopt a two-stage training paradigm initialized from Qwen3-32B: (1) end-to-end Supervised Fine-Tuning (SFT), followed by (2) module-specific SFT and Reinforcement Learning (RL).

3.4.1 End-to-End Supervised Fine-Tuning

Stage 1 establishes general capabilities for multi-step analytical tasks using 32k end-to-end trajectories covering Data Source, NL2SQL, and Code Agent modules. Standard SFT resolves initial deficiencies in problem decomposition, code generation, and format adherence.

3.4.2 Module-Specific SFT and RL Optimization

Stage 1 revealed persistent SQL-specific weaknesses: interpreting complex schemas, applying multi-level constraints, and utilizing external financial knowledge. We therefore target the Sqltool module for specialized optimization.

SFT Cold-Start and Data Selection We first perform SFT on 17k complex Data Source and

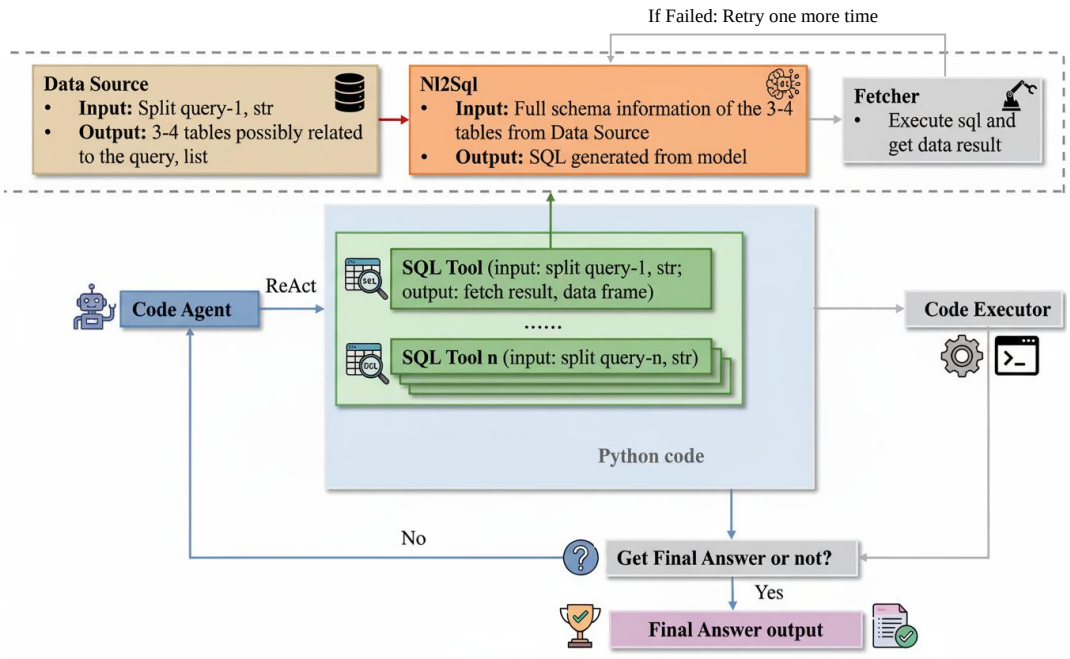


Figure 2: Generating Pipeline

NL2SQL pairs grounded in financial constraints. Using $pass@8$ rejection sampling, we identify trajectories for RL.

Figure 3(a) shows a **bi-modal distribution**: samples either fail completely (0/8) or pass perfectly (8/8). Both extremes suffer from *gradient sparsity*. We retain intermediate scores (0.1–0.9), yielding 4k high-quality trajectories: 2,937 for NL2SQL ($\mu = 0.52$) and 1,191 for Data Source ($\mu = 0.56$)—a "Goldilocks zone" for informative RL gradients.

Rule-Based Reward Design We use execution-based rewards instead of LLM-as-a-judge. For NL2SQL, binary reward R_G checks execution equivalence; for Data Source, R_D checks set matching.

$$R = \begin{cases} R_G, & \tau = \text{nl2sql} \\ R_D, & \tau = \text{data_source} \end{cases} \qquad R_G = \mathbb{1}[\text{Exec}(y) = \text{Exec}(y^*)], \quad R_D = \mathbb{1}[\text{Set}(y) = \text{Set}(y^*)] \tag{1}$$

GSPO Algorithm We adapt Group Relative Policy Optimization with three modifications: (1) sequence-level importance sampling with truncation, (2) no KL penalty, and (3) tight clipping.

$$\mathcal{L}^{\text{GSPO}} = -\mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G} \min\left(\tilde{r}_i \hat{A}_i, \text{clip}(\tilde{r}_i, 1-\epsilon, 1+\epsilon)\hat{A}_i\right)\right] \tag{2}$$

$$\tilde{r}_i = \exp\left(\frac{1}{|o_i|}\sum_{t=1}^{|o_i|} \log \frac{\pi_\theta(o_{i,t})}{\pi_{\text{old}}(o_{i,t})}\right) \tag{3}$$

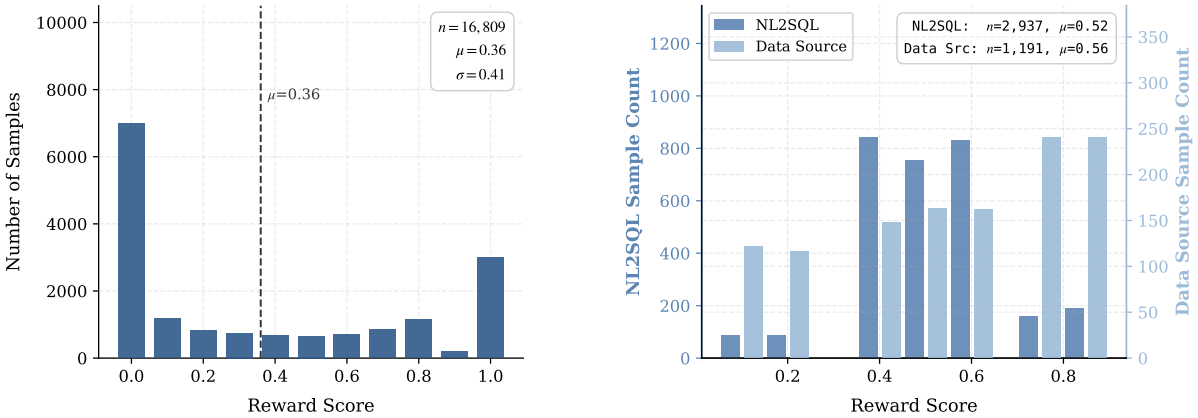
We filter samples with $\tilde{r}_i > 2$ and use $\epsilon \in [3 \times 10^{-4}, 4 \times 10^{-4}]$, ensuring stable updates without policy collapse.

4 Experiment

We evaluate our two-stage training paradigm on our internal financial benchmark (Self-built) and the BIRD dev dataset. Table 3 summarizes execution accuracy across training phases.

Stage 1 SFT improves over Qwen3-32B baseline by +12.98% on Self-built, establishing foundational multi-module capabilities. Stage 2 RL further pushes performance to 89.72% on Self-built and 49.74% on BIRD, validating rejection sampling-based RL for complex SQL generation.

Domain-Specific Data Necessity Existing open-source datasets (Spider, OmniSQL) show limited efficacy in financial scenarios. Adding 14k samples from Spider or OmniSQL yields only +2.36% and +2.49% gains on Self-built, with OmniSQL achieving only 81.48% on Bank and 63.63% on Other tasks—far below our approach. This severe domain mismatch confirms that general datasets fail to capture financial schema constraints, accounting logic, and metric formulas.



(a) Distribution of Raw Rollout Samples

(b) Distribution of Filtered Subsets

Figure 3: Reward score distributions during data selection. (a) Raw scores of 16.8k trajectories show bi-modal distribution. (b) Filtered subsets focus on medium-score range for dense RL signal.

Table 3: **Ablation study of training stages and data composition.** Execution Accuracy (Ex Acc.) is reported as the mean of three independent runs (Avg@3). We compare our specialized *Self-synthesis* data against established open-source datasets including Spider (Yu et al., 2018) and OmniSQL (Li et al., 2025). The **Stage 2-RL** represents the final model optimized via GSPO on 4k rejection-sampled trajectories.

Training Stage & Data	**Self-built Ex Acc. (Avg@3)**			**Bird Dev Ex Acc. (Avg@3)**			
	Bank	Others	Total	Simple	Moderate	Complex	Total
(baseline) Qwen3-32b	68.42	40.00	61.84	42.09	44.90	42.99	43.02
(stage 1) End-to-end 32k SFT	77.47	66.15	74.82	42.92	45.26	43.45	43.68
(stage 2-SFT) Stage 1 + Spider 14k	79.00	71.21	77.18	44.68	48.78	48.51	46.28
(stage 2-SFT) Stage 1 + Ominisql 14k	81.48	63.63	77.31	46.63	51.29	47.82	48.16
(stage 2-SFT) Stage 1 + Self-synthesis 17k	**85.65**	**82.82**	**84.99**	**48.03**	**51.36**	**47.36**	**48.98**
(stage 2-RL) Stage 2 + Rollout 4k	**90.74**	**86.36**	**89.72**	**48.86**	**52.37**	**46.90**	**49.74**

In contrast, our Self-synthesis 17k dataset, grounded in financial knowledge, achieves 84.99% on Self-built—a +7.68% improvement over OmniSQL. This domain-specific tuning generalizes well: on BIRD dev, Self-synthesis reaches 48.98%, surpassing Spider (+2.7%) and OmniSQL (+0.82%).

RLHF Bridges the Last Mile While SFT provides strong cold-start, RL is essential for execution correctness. Optimizing on 4k high-signal samples boosts Self-built "Others" accuracy from 82.82% to 86.36%, demonstrating RL's potency in resolving ambiguous logic and enforcing complex financial constraints missed by supervised learning.

5 Conclusions

In this work, we addressed the fundamental gap between academic Text-to-SQL research and the rigorous requirements of enterprise financial analytics. We demonstrated that the primary hurdles—intricate schema mapping, domain-specific logic, and temporal constraints—cannot be fully resolved by simply scaling general-purpose SFT.

Our proposed framework offers a three-fold solution: (i) a **high-fidelity synthesis pipeline** that provides the model with "industrial-grade" exposure to complex fact/auxiliary table structures; (ii) a **modular Agentic SQL Pipeline** that prioritizes holistic Data Source localization over granular schema linking, ensuring the model maintains macro-level awareness of database relations; and (iii) a **two-**

stage GSPO training paradigm that utilizes rejection sampling to focus reinforcement learning on the most informative learning gradients.

Experimental results validate our approach, showing that our specialized "SQL-first" synthesis outperforms established open-source datasets (Spider/OmniSQL) in domain-specific tasks while maintaining strong generalization on cross-domain benchmarks like BIRD. Specifically, the success of our RL phase in closing the "last mile" of execution correctness underscores the necessity of moving toward execution-verified, feedback-driven optimization for mission-critical financial applications. Future work will explore the integration of more diverse agentic tools, such as automated visual analytics, to further extend the model's end-to-end report generation capabilities.

6 Limitations

Despite its performance, our framework has limitations. First, the **computational cost** of the two-stage training, particularly the multi-agent GSPO phase, is higher than standard SFT. Second, our synthesis pipeline, while highly realistic, still relies on **LLM-based attribute specification**, which may inherit subtle biases from the base model. Third, the current **Agentic Pipeline** assumes a relatively stable database schema; in environments where schemas evolve rapidly in real-time, the Data Source module would require more frequent fine-tuning or a more adaptive retrieval mechanism. Finally, while we use LLM-as-a-judge for final report evaluation, the potential for **inconsistent scoring** on highly nuanced financial attributions remains an area for further refinement.

References

Thanh Dat Hoang, Thanh Tam Nguyen, Thanh Trung Huynh, Hongzhi Yin, and Quoc Viet Hung Nguyen. 2025. Scaling text2sql via llm-efficient schema filtering with functional dependency graph rerankers. *arXiv e-prints*, pages arXiv–2512.

Minghang Deng, Ashwin Ramachandran, Canwen Xu, Lanxiang Hu, Zhewei Yao, Anupam Datta, and Hao Zhang. 2025. Reforce: A text-to-sql agent with self-refinement, consensus enforcement, and column exploration. *arXiv preprint arXiv:2502.00675*.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv e-prints*, pages arXiv–2307.

Zhifeng Hao, Qibin Song, Ruichu Cai, and Boyan Xu. 2025. Text-to-sql as dual-state reasoning: Integrating adaptive context and progressive generation. *arXiv e-prints*, pages arXiv–2511.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 337–353.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, and 1 others. 2025. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proceedings of the VLDB Endowment*, 18(11):4695–4709.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023. 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*, 36:42330–42357.

Yifu Liu, Yin Zhu, Yingqi Gao, Zhiling Luo, Xiaoxia Li, Xiaorong Shi, Yuntao Hong, Jinyang Gao, Yu Li, Bolin Ding, and 1 others. 2026. Xiyan-sql: A novel multi-generator framework for text-to-sql. *IEEE Transactions on Knowledge and Data Engineering*.

Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. In *NeurIPS 2024 Third Table Representation Learning Workshop*.

Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Sercan O Arik. 2024. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *arXiv e-prints*, pages arXiv–2410.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in neural information processing systems*, 36:36339–36348.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan O Arik. Reasoning-sql: Reinforcement learning with sql tailored partial rewards for reasoning-enhanced text-to-sql. In *Second Conference on Language Modeling*.

Lei Sheng and Xu Shuai Shuai. 2025. Csc-sql: Corrective self-consistency in text-to-sql via reinforcement learning. In *Proceedings of the 14th International Joint Conference on Natural Language Processing and the 4th Conference of the Asia-Pacific Chapter of the Association for Computational Linguistics*, pages 1473–1496.

Vladislav Shkapenyuk, Divesh Srivastava, Theodore Johnson, and Parisa Ghane. 2025. Automatic metadata extraction for text-to-sql. *arXiv e-prints*, pages arXiv–2505.

Khushboo Thaker and Yony Bresler. 2025. Knowledge distillation with structured chain-of-thought for text-to-sql. *arXiv e-prints*, pages arXiv–2512.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and 1 others. 2025a. Mac-sql: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557.

Pengfei Wang, Baolin Sun, Xuemei Dong, Yaxun Dai, Hongwei Yuan, Mengdie Chu, Yingqi Gao, Xiang Qi, Peng Zhang, and Ying Yan. 2025b. Agentar-scale-sql: Advancing text-to-sql through orchestrated test-time scaling. *arXiv e-prints*, pages arXiv–2509.

Ziyang Wang, Yuanlei Zheng, Zhenbiao Cao, Xiaojin Zhang, Zhongyu Wei, Pei Fu, Zhenbo Luo, Wei Chen, and Xiang Bai. 2025c. Autolink: Autonomous schema exploration and expansion for scalable schema linking in text-to-sql at scale. *arXiv e-prints*, pages arXiv–2511.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, and 1 others. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 conference on empirical methods in natural language processing*, pages 3911–3921.

Yuxin Zhang, Meihao Fan, Ju Fan, Mingyang Yi, Yuyu Luo, Jian Tan, and Guoliang Li. 2025. Reward-sql: Boosting text-to-sql via stepwise reasoning and process-supervised rewards. *arXiv e-prints*, pages arXiv–2505.

A Appendix

To ensure the reliability and reproducibility of the evaluation, we performed a rigorous refinement of the benchmark. This process focuses on two core objectives: (1) eliminating ambiguity through domain-specific data logic and (2) reducing scoring variance by minimizing Ground Truth (GT) complexity.

A.1 Standardization of Financial Data Logic

We identified that model inconsistencies often stem from a lack of domain-specific knowledge, particularly regarding financial accounting principles. We standardized the data processing rules into two primary categories as detailed in Table 4.

A.2 Evaluation Rubric and GT Minimization

To mitigate the noise introduced by long-form answers in "LLM-as-a-judge" scenarios, we implemented a Ground Truth Minimization strategy. By stripping away redundant context and focusing on core facts, we reduce the "hallucination" risk during the scoring phase.

Scoring Rubric We utilize a three-tier rubric for human and automated evaluation:

- **1.0 Point (Correct):** The response is logically sound, fully consistent with the GT, and covers all core factual nodes.
- **0.5 Points (Partially Correct):** The response identifies some core nodes but contains minor calculation errors, slight factual deviations, or omissions.
- **0 Points (Incorrect):** The response misses core factual nodes, exhibits logical fallacies, or contradicts the GT.

Case Study: GT Minimization An example of our GT refinement is shown in Table 5. By focusing on the final calculation result rather than repeating the entire context, we ensure the scoring model focuses on the accuracy of the answer.

B Appendix

The design of our two-stage training paradigm (Section 3.4) was empirically driven by a preliminary diagnostic analysis. By identifying specific failure modes in the initial model, we tailored Stage 1 to address general analytical capabilities and Stage 2 to resolve deep domain-specific logic.

B.1 Preliminary Error Analysis (Guidance for Stage 1)

Prior to the Supervised Fine-Tuning (SFT) phases, we conducted a manual error analysis on part of failed interaction trajectories from our internal financial benchmarks (`GZNS`, `YX`, and `CSNS`). The goal was to categorize the fundamental weaknesses of the base model. The distribution of these initial error types is summarized in Table 6.

As shown in the analysis, the failure modes were not isolated to a single module but distributed across the entire analytical pipeline, encompassing SQL Retrieval (20.7%), Python Execution (18.0%), Agent Planning (17.1%), and Output Formatting

Category	Data Type	Processing Logic & Domain Constraints
1. Temporal Logic	Stock Data	Refers to values at a specific point in time (e.g., Balance Sheets). **Constraint:** These values cannot be summed across time intervals (e.g., adding monthly debt figures to get an annual total is invalid).
	Flow Data	Refers to changes over a period (e.g., Income Statements). **Logic:** If the database stores cumulative year-to-date (YTD) values, the periodic value (e.g., monthly) must be derived by subtracting the previous period's accumulation.
2. Business Constraints	Data Filtering	Certain tables contain overlapping entries (e.g., personal loan tables including corporate data). **Constraint:** Models must apply explicit filters (e.g., `WHERE category='personal'`) to ensure a unique and correct retrieval.

Table 4: Standardized logic for financial data processing to resolve domain-specific ambiguity.

Question	*What is the ratio of active retail deposits in Baiyun Branch to total retail deposits as of mid-2023?*
Original GT	As of June 30, 2023, the active retail deposits in Baiyun Branch were 3,637,920,000 RMB, and the total retail deposits were 5,596,800,000 RMB. The ratio of active retail deposits to total retail deposits is 0.6500 (65.00%).
Refined GT	**The ratio is 0.6500, or 65.00%.**

Table 5: Comparison between original and minimized Ground Truth. The minimized version reduces evaluator distraction.

(7.2%). This broad spectrum of deficiencies indicated that the base model lacked the foundational competency required for multi-step interactions. Consequently, we designed Stage 1 as an end-to-end Supervised Fine-Tuning (SFT) phase. Rather than targeting specific modules, this holistic approach aims to comprehensively establish the model's general capabilities, including problem decomposition, basic analytical programming, and strict format adherence.

B.2 Case Studies: NL2SQL Challenges in Financial Scenarios

We categorize the observed failures in Stage 1 into three main types of "Domain Misunderstandings." Stage 2 specifically addresses these through knowledge-augmented SFT and RL.

Case A: Failure to Execute Statistical Calculation Logic. Financial databases frequently record flow data (e.g., revenue, sales volume) as cumulative Year-to-Date (YTD) values. To query data for a specific month, the model must subtract the previous month's cumulative total from the current month's total. As detailed in Table 7, even when the system explicitly provides this calculation rule in the supplementary prompt, the Stage 1 model ignores the instruction. It defaults to a naive direct query instead of constructing the required multi-step self-join and subtraction. This inability to translate complex accounting logic into SQL operations motivated the specialized optimization in Stage 2.

Case B: Schema Ambiguity and Neglect of Enumerated Values. Enterprise financial databases inherently contain multiple tables with semantically similar column names (e.g., *Organization Code* vs. *Branch Code*). To accurately map a natural language query to the correct table, the model must cross-reference the requested entity with the enumerated values (Enums) provided in the schema metadata. As demonstrated in Table 8, the Stage 1 model exhibits a shallow reliance on semantic similarity matching. When asked for a specific branch, it maps the query to the intuitively named `dim_branch_info` table, completely ignoring the explicit Enum hints indicating that specific operational branches (e.g., 'Puxi Branch') only exist in the `cust_info` table. This failure to leverage

Error Category	GZNS	YX	CSNS	Total (%)
SQL Retrieval Errors	0.5	0.5	10.5	**20.7%**
Agent Planning Failures	4.0	0.0	5.5	**17.1%**
Python Code Errors	3.0	7.0	0.0	**18.0%**
Data Unit Inconsistencies	0.0	0.0	1.5	2.7%
Output Formatting Errors	0.0	0.0	4.0	7.2%
Ambiguous Queries	0.5	0.5	2.0	5.4%
Ground Truth Noise	2.0	0.0	6.0	14.4%
Evaluator Misjudgment	2.5	4.5	1.0	14.4%

Table 6: Initial distribution of error types that **guided** our two-stage training strategy. *Includes field name inconsistencies, timestamp recognition errors, and ratio calculation flaws.

schema metadata for disambiguation necessitates the knowledge-injected RL optimization in Stage 2.

Case C: Failure to Incorporate Implicit Business Constraints. Financial queries often operate under implicit default constraints—rules that are not explicitly stated in the user's question but are essential for business accuracy. For instance, in our bank's reporting logic, if a query mentions no specific branch, it must default to the "Total Bank" (Headquarters) level data. As shown in Table 9, the Stage 1 model correctly identifies the temporal and thematic elements of the query but fails to integrate the implicit filtering rule specified in the supplementary metadata. This leads to an over-broad query that aggregates data across all branches rather than filtering for the required headquarters-level summary. We utilize Stage 2 to reinforce the model's ability to reason with these "if-then" business logic constraints.

Component	Content
System Prompt (Supplementary)	**Rules for Flow Data Calculation:** If the database stores cumulative values, obtaining data for a specific period requires calculating the difference. For example, to get data for Month X, you must use: *(Cumulative value at the end of current month) - (Cumulative value at the end of previous month).*
User Query	Query the sales volume of live pigs for July 2024.

Model Output (Stage 1)	**[Incorrect]** The model ignored the subtraction rule and directly queried the target date.
	`SELECT sales_volume`
	`FROM agriculture_sector_v2`
	`WHERE date = '2024-07-31';`

Target SQL (Correct)	**[Correct]** The model should apply a self-join to subtract June's accumulation from July's.
	`SELECT t1.sales_volume - COALESCE(t2.sales_volume, 0) AS monthly_sales`
	`FROM (SELECT sales_volume FROM agriculture_sector_v2`
	`WHERE date = '2024-07-31') t1`
	`LEFT JOIN (SELECT sales_volume FROM agriculture_sector_v2`
	`WHERE date = '2024-06-30') t2 ON 1=1;`

Table 7: Detailed Failure Case A. The Stage 1 model fails to follow explicit prompt instructions regarding cumulative data calculation, highlighting a critical deficiency in complex SQL construction (e.g., self-joins and arithmetic operations).

Component	Content
Database Schema (Provided in Prompt)	**Table 1:** `cust_info` (Corporate Customer Basic Info) - `ORGID`: Org Code (Enum: '103200', '103201') - `Attr_Org`: Affiliated Branch (**Enum: 'Puxi Branch'**, 'Shenzhen Branch') **Table 2:** `dim_branch_info` (Branch Parameter Table) - `BRANCHID`: Branch Code (Enum: '101999', '101200') - `BRANCHID_name`: Branch Name (**Enum: 'Total Bank'**, 'Local Institution')
User Query	Query the organization code of the Puxi Branch.

Model Output (Stage 1)	**[Incorrect]** The model mistakenly chose Table 2 due to the semantic overlap of the word "Branch", hallucinating the value 'Puxi Branch' into a column that only contains high-level aggregates.
	`SELECT BRANCHID`
	`FROM dim_branch_info`
	`WHERE BRANCHID_name = 'Puxi Branch';`

Target SQL (Correct)	**[Correct]** The model must deduce from the Enums that 'Puxi Branch' is a valid value only in Table 1's `Attr_Org` column, thus retrieving the corresponding `ORGID`.
	`SELECT ORGID`
	`FROM cust_info`
	`WHERE Attr_Org = 'Puxi Branch';`

Table 8: Detailed Failure Case B. The Stage 1 model fails to disambiguate semantically similar columns across different tables because it neglects the explicit enumerated values (Enums) provided in the schema metadata.

Component	Content
Business Rule (Supplementary)	**Default Scope Constraint:** If no specific institution or branch is mentioned in the query, the model MUST default to querying data for the **"Total Bank / Headquarters / Group"** level. *(Note: In dim_branch_info, the 'Total Bank' ID is '101999'.)*
User Query	Query the loan balance for each business line as of September 30, 2023.

Model Output (Stage 1)	**[Incorrect]** The model identifies the date and table but neglects the implicit branch filter, resulting in an aggregated total for all sub-branches instead of the specified "Total Bank" summary.
	`SELECT I.LINE, li.LINE_NAME, SUM(I.VAL) AS loan_balance`
	`FROM idw_t_loan I LEFT JOIN dim_line_info li ON I.LINE = li.LINE`
	`WHERE I.STATDATE = '2023-09-30 00:00:00'`
	`GROUP BY I.LINE, li.LINE_NAME;`

Target SQL (Correct)	**[Correct]** The model correctly incorporates the implicit constraint by adding the default branch filter `'101999'` to the `WHERE` clause.
	`SELECT I.LINE, li.LINE_NAME, SUM(I.VAL) AS loan_balance`
	`FROM idw_t_loan I LEFT JOIN dim_line_info li ON I.LINE = li.LINE`
	`WHERE I.STATDATE = '2023-09-30 00:00:00' AND I.BRANCHID = '101999'`
	`GROUP BY I.LINE, li.LINE_NAME;`

Table 9: Detailed Failure Case C. The Stage 1 model fails to account for implicit business logic (default institution filters) provided in the supplementary instructions, emphasizing the need for advanced reasoning training in Stage 2.