# ENHANCING ROBUSTNESS OF SMALL-SCALE MODELS FOR TEXT-TO-SQL VIA SEMANTIC ENHANCEMENT AND NOISE-AWARE METHODS

**Anonymous authors**
Paper under double-blind review

## ABSTRACT

With the advancement of large language model technology, Text-to-SQL techniques have achieved significant progress and are widely applied in scenarios such as search engines. However, current small-scale models still face challenges, particularly poor robustness, especially when the database schema or user natural language queries lack clear semantic descriptions, resulting in decreased SQL generation accuracy. Although large-scale models demonstrate stronger robustness and higher accuracy, their high computational and deployment costs limit their use in private deployment settings. To address these issues, this paper proposes a reverse semantic enhancement method that leverages prior knowledge such as existing database values to enrich the semantic information of the database schema and user queries. Furthermore, to mitigate the noise introduced during semantic enhancement, a noise-aware training approach is designed to critique noisy data and identify effective semantic information, thereby improving the model's resistance to interference. Extensive experiments on the SPIDER and BIRD datasets demonstrate the effectiveness of the proposed methods.

## 1 INTRODUCTION

The Text-to-SQL task aims to transform natural language queries from users into executable SQL statements, thereby enabling users without specialized SQL expertise to interact with databases intuitively Katsogiannis-Meimarakis & Koutrika (2023). With the rapid advancement of Large Language Models (LLMs)Zhao et al. (2023), the accuracy of Text-to-SQL has improved significantly, leading to its widespread adoption in practical applications such as Retrieval-Augmented Generation (RAG)Lewis et al. (2020). However, current state-of-the-art Text-to-SQL methods primarily rely on models with massive parameter scales. In contrast, small-scale models still suffer from poor robustness; their output stability is highly susceptible to the semantic information present in the input text. When the input is semantically impoverished, the model's output often deviates substantially from the expected result.

As illustrated in Figure 1, semantic poverty in current Text-to-SQL tasks manifests primarily across two dimensions: the database schema and the user query. On the one hand, regarding the Schema (Figure 1(a)), column naming is often non-standardized and ambiguous. For instance, a column recording the establishment time is vaguely named `Creation`. When a user queries about a specific "year," the model fails to establish a semantic connection between the two, erroneously hallucinating a non-existent `year` column. On the other hand, regarding the Query (Figure 1(b)), users frequently employ domain-specific vocabulary or advanced synonyms. In this example, the user employs the term *faculties* to refer to teachers, which is not a conventional expression in the database. Furthermore, the database schema only contains a `teacher` table, with no direct correspondence to `faculties`. This creates a severe semantic gap between the user query and the schema. For small-scale models, it is exceptionally challenging to automatically identify and bridge this semantic gap, inevitably leading to SQL generation errors. In current Text-to-SQL tasks, semantic poverty—such as irregular field naming and ambiguous user queries—is a pervasive issue. These factors can all contribute to a decline in model robustness and severely compromise the stability of the output.



Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

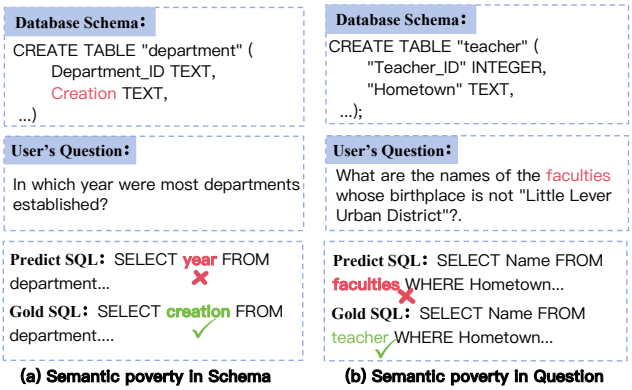



Figure 1: Examples of dual semantic poverty in Text-to-SQL

Extensive research has been dedicated to improving model robustness in Text-to-SQL tasks. For instance, benchmarks such as Spider-SynGan et al. (2021), Spider-RealisticDeng et al. (2020), and Dr.SpiderChang et al. (2023) have explored the impact of various perturbation strategies. These works enhance robustness through adversarial samples Gan et al. (2021); Deng et al. (2020) and weakly supervised training frameworks Chang et al. (2023). Additionally, Pi et al. (2022) improves robustness by constructing tabular adversarial samples, while Shen et al. (2023) designs a sequential feature enhancement method. Other efforts have leveraged data augmentation Li et al. (2024a; 2025); Cai et al. (2025) and novel Text-to-SQL framework designs Qin et al. (2024); Liu et al. (2024a); Cao et al. (2024) to further this objective. While these methods have effectively improved robustness, they generally overlook the interference caused by the semantic poverty of the schema and user queries. Consequently, the challenge of mitigating the impact of semantic poverty on model robustness remains unresolved.

To address this problem, a straightforward solution is to apply semantic enhancement to both the database schema and user queries. This improves the model's understanding and reduces the impact of semantic gaps. However, this approach faces two main challenges:

**(1) How to effectively enhance the semantics of database schemas and user questions?** Small-scale models have limited capacity and cannot achieve effective semantic enhancement through simple prompting. Large, high-performance models (e.g., DeepSeekGuo et al. (2024)) are too computationally expensive for small-scale private deployment. Thus, it is important to train small models with inherent semantic inference abilities. We find that database values, user questions, and past NL-SQL pairs contain latent semantic information that can supplement the schema. For example, if a column has values like 'USA', we can infer it refers to 'Country' even when the column name is unclear. Similarly, SQL queries provide clues about user intent. We identify two levels of hints derived from SQL: clause-level (e.g., sub-SQL linked to sub-questions) and entity-level (e.g., mapping question entities to columns). Based on these insights, we propose a reverse semantic inference data synthesis method to generate semantic hints for schemas and questions, enabling small models to learn schema enrichment and hint generation for new Text-to-SQL tasks.

**(2) How to perceive and handle noise generated during the semantic enhancement process?** While semantic enhancement enriches schema and query information and improves SQL accuracy, incorrect enhancements can cause harm. Therefore, the Text-to-SQL model needs to be noise-aware. To achieve this, we propose a noise-aware training method integrated with the GRPO (Group Relative Policy Optimization)Shao et al. (2024) algorithm. By injecting semantically similar but incorrect noise into the training data, the model is rewarded for correctly identifying and correcting noisy information. This trains the model to develop strong noise detection and correction abilities. In summary, our contributions are:

- We propose a novel reverse semantic inference method to synthesize semantic information for the database schema and user queries, utilizing the synthesized data to train small-scale models with semantic inference capabilities.
- We introduce a novel noise-aware Text-to-SQL approach. When generating SQL using the semantically enhanced schema and user queries, the model simultaneously perceives potential semantic noise, selectively utilizing only the correct semantic information to assist in accurate SQL generation.
- We conduct extensive experiments to demonstrate the effectiveness of our proposed methods, significantly improving the robustness of small-scale models in Text-to-SQL tasks.

## 2 Related Work

**Text-to-SQL**: Driven by the rapid advancement of Large Language Models (LLMs) Li et al. (2024b), Text-to-SQL has transitioned from traditional Seq2Seq networks Liu et al. (2024b) to highly accurate LLM-based paradigms. Current methodologies primarily fall into two categories: such as Codes Li et al. (2024a), Omni-SQL Li et al. (2025), and semantic enhancement frameworks such as SEA-SQL Li et al. (2026), which improve schema-query alignment through adaptive semantic refinement; and training-free approaches, which achieve state-of-the-art performance via Chain-of-Thought (CoT) Wei et al. (2022) prompting (e.g., C3SQL Dong et al. (2023), DIN-SQL Pourreza & Rafiei (2024)). Additionally, multi-agent frameworks like MAC-SQL Wang et al. (2024) and SQLFixAgent Cen et al. (2024) have been proposed to further refine SQL generation.

**Robustness in NLP**: Enhancing robustness remains a critical challenge in NLP Wang et al. (2022), largely addressed through adversarial training. Adversarial examples are typically constructed at three granularities: character-level (e.g., character swapping via HotFlip Ebrahimi et al. (2018)), word-level (e.g., synonym substitution and embedding perturbations Alzantot et al. (2018); Ren et al. (2019)), and sentence-level (e.g., paraphrasing or adding distractor sentences like AddSent Jia & Liang (2017)).

**Robustness in Text-to-SQL**: Current research on Text-to-SQL robustness mainly focuses on building benchmarks and designing robustness-oriented training methods. Benchmarks like Spider-Syn Gan et al. (2021), Spider-Real Deng et al. (2020), and Dr.Spider Chang et al. (2023) evaluate model weaknesses against synonym substitutions, missing schema information, and structural changes. To address these challenges, prior work has proposed methods such as table-side perturbation defense Pi et al. (2022), sequential feature enhancement Shen et al. (2023), three-stage robustness training Gao et al. (2022), schema evolution-based robustness improvementZhang et al. (2026), and data-augmented schema linking (Solid-SQL Liu et al. (2024a)). More recent work, like JOLT-SQL Song et al. (2025), enhances robustness under noisy schemas by using confusion-aware noisy schema sampling during training.

## 3 Methodology

As shown in Figure 2, our framework has three main phases:

***(1) Semantic Enhancement via Reverse Semantic Inference (Top-left of Fig 2):*** This phase extracts hidden associations from historical queries, example values in columns, database instances, and links between SQL statements and user questions. These associations are used to build a high-quality training set for training the Semantic Synthesis Generator (SynGen).

***(2) Noise-Aware Text-to-SQL Method (Top-right of Fig 2):*** We inject semantically similar noise into the enhanced schemas and user questions. A noise-aware reward mechanism is applied to improve the model's robustness against interference.

***(3) Integrated Inference Flow (Bottom of Fig 2):*** In this phase, SynGen is combined with the noise-aware Text-to-SQL model (Noisy-SQL) to achieve robust, end-to-end SQL generation.

### 3.1 Semantic Enhancement via Reverse Semantic Inference

As shown in the top-left of Figure 2, this section explains how to generate semantically enhanced data for the Text-to-SQL training set. Using this data, we train the SynGen model to enrich semantics in new Text-to-SQL inputs.

**Schema Enrichment:** Database column names are often abstract, with their true meaning hidden in data values and past queries. For example, in the top-left of Figure 2, the `creation` column in the `department` table contains the value "1789". Together with the historical query "In which year were most departments established?" and its SQL `SELECT creation...`, the model can infer that the `creation` column represents the "Year the department was created". Such contextual information is essential when column names are vague or generic. Based on this, we use sampled values and queries to create prompts that guide schema enrichment. Formally, given a database schema $S$, example values $V$, and historical query pairs $(Q, Y)$, the task is to generate enhanced

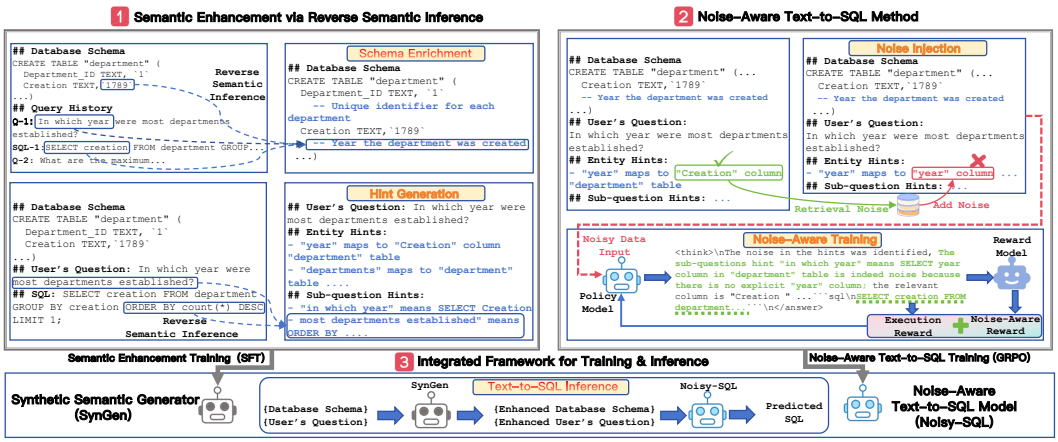



Figure 2: Illustration of the Semantic Enhancement and Noise-Aware Text-to-SQL Framework.

descriptions $S^*$:

$$S^* = f(S, V, Q, Y) \tag{1}$$

where $f$ is a Large Language Model (LLM).

**Hint Generation:** User queries often lack sufficient semantic information, so enhancing their semantics is also important. We identify two levels of correspondence between natural language queries and SQL: entity-level and sub-question-level. Entity-level relates to basic schema elements like columns, tables, and values, while sub-question-level corresponds to SQL query clauses. Based on this, we apply reverse semantic inference to queries at both levels: **(1) Entity-level Enhancement** ($H_{ent}$): Maps nouns in the query $Q$ to table or column names in the schema $S$. **(2) Sub-question-level Enhancement** ($H_{sub}$): Breaks down complex queries into sub-questions linked to specific SQL clauses. Given manual examples $E$, the enhanced query $Q^*$ is defined as:

$$Q^* = \{Q, H_{ent}, H_{sub}\} = f(E, S, Q, Y) \tag{2}$$

As shown in the top-left of Figure 2, we use historical queries and SQL statements with a few-shot LLM approach to guide entity- and clause-level inference. For example, from the SQL, "year" maps to the `creation` column, and "most departments established" corresponds to `ORDER BY count(*) DESC LIMIT 1`, providing multi-level semantic hints for the query.

**Training the Semantic Synthesis Generator (SynGen):** After obtaining the enhanced schema $S^*$ and question $Q^*$, we utilize this synthetic data to train a specialized Semantic Synthesis Generator (SynGen). Unlike the data synthesis phase, the Ground Truth SQL ($Y$) is excluded during the formal training of SynGen to ensure the model relies solely on the original schema $S$ and question $Q$ during inference. Specifically, given the training dataset $\mathcal{D}_{syn} = \{(S, Q, S^*, Q^*)_i\}_{i=1}^{N}$, the training objective is to minimize the following negative log-likelihood loss:

$$\mathcal{L}_{syn}(\theta_{syn}) = -\sum_{i=1}^{N} \log P(S^*, Q^* | S, Q; \theta_{syn}) \tag{3}$$

where $\theta_{syn}$ represents the model parameters. Through this training phase, SynGen learns the capability to autonomously infer latent semantics from the raw schema and natural language questions.

### 3.2 Noise-Aware Text-to-SQL Method

While semantic enhancement aids SQL generation, incorrect semantic hints can mislead the model. We address this using a reinforcement learning-based noise-aware training method (Figure 2, top-right).

**Noise Injection:** To build the model's noise recognition capabilities, we inject hard negative noise into the enhanced data (Section 3.1). For any ground-truth comment $c \in S^*$ or hint $h \in Q^*$, we use a

vector encoder $\phi(\cdot)$ to retrieve the most semantically similar non-target item as noise:

$$\tilde{c} = \arg \max_{c' \in \mathcal{C}, c' \neq c} \text{sim}(\phi(c), \phi(c')) \tag{4}$$

For instance, a correct hint mapping "year" to the `creation` column might be replaced with a semantically similar but incorrect mapping to disrupt the query.

**Noise-Aware Training:** We optimize the policy model $\pi_\theta$ using **GRPO**. Unlike PPOSchulman et al. (2017), GRPO estimates the baseline by sampling a group of responses without requiring a separate value network.

For each input $q$, we sample $G$ reasoning trajectories and SQL outputs $\{o_i\}_{i=1}^G$ from the old policy $\pi_{\theta_{old}}$. We calculate the normalized group-relative advantage $\hat{A}_i$ using the total reward $R_{total}^{(i)}$ of each output:

$$\hat{A}_i = \frac{R_{total}^{(i)} - \text{mean}(\{R_{total}^{(1)}, \ldots, R_{total}^{(G)}\})}{\text{std}(\{R_{total}^{(1)}, \ldots, R_{total}^{(G)}\})}$$

The policy maximizes the GRPO objective:

$$\begin{aligned} \mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{q,\{o_i\}_{i=1}^G} \Bigg[ \frac{1}{G} \sum_{i=1}^{G} \Bigg( \min \Big( \rho_i(\theta) \hat{A}_i, \\ \text{clip}(\rho_i(\theta), 1-\epsilon, 1+\epsilon) \hat{A}_i \Big) - \lambda \mathbb{D}_{KL}(\pi_\theta \| \pi_{ref}) \Bigg) \Bigg] \end{aligned} \tag{5}$$

where $\rho_i(\theta)$ is the probability ratio, $\epsilon$ clips policy updates, and $\lambda$ penalizes KL divergence from the reference model $\pi_{ref}$.

To simultaneously improve noise perception and SQL accuracy, **Noisy-SQL** is trained with a multi-dimensional reward:

$$R_{total} = \alpha \cdot R_{exe} + \beta \cdot R_{fmt} + \gamma \cdot R_{noise}$$

**(1) Execution Reward** ($R_{exe}$): Ensures logical correctness by matching the generated SQL's execution result against the gold standard.

**(2) Format Reward** ($R_{fmt}$): Enforces strict structural compliance for both the CoT reasoning and the SQL code block (configured similarly to DeepSeek-R1Guo et al. (2025)).

**(3) Noise-Aware Reward** ($R_{noise}$): Since reasoning trajectories are diverse, we use `Qwen2.5-72B-Instruct` as a Reward Model (RM) to automatically evaluate noise handling:

$$R_{noise} = RM(\text{Input}, \text{Thought_Process}, \text{Noise_Labels})$$

The RM assigns higher scores when the model explicitly identifies and corrects noisy hints within its `<think>` reasoning trajectory.

**Integrated Inference Workflow:** At inference, **SynGen** enhances the raw schema $S$ and query $Q$ into $\{S^*, Q^*\}$. The **Noisy-SQL** model then processes these inputs, using CoT to proactively filter semantic noise and generate a robust predicted SQL $\hat{Y}$.

# 4 Experiments

## 4.1 Experimental Setup

**Datasets:** We conduct experiments on BIRD and Spider datasets, which are popular benchmarks for Text-to-SQL tasks. For robustness evaluation, we use the Spider-syn (Gan et al., 2021) and Spider-Real (Deng et al., 2020) variants. Detailed descriptions of these datasets are provided in Appendix A.1.

**Evaluation Metrics:** We use execution accuracy to evaluate SQL generation, VES to assess SQL execution efficiency, and exact match (EM) to measure improvements in SQL component accuracy. Detailed descriptions of these metrics are provided in Appendix A.2.

**Implementation Details:** Our experiments primarily use Qwen2.5-Coder-3B-Instruct (Hui et al., 2024). To verify the generality of our approach, we also evaluate on models of different scales, including the 1.5B versions of Qwen, as well as other models such as Llama-3.1-3B-Instruct (Grattafiori et al., 2024). Additional implementation details can be found in Appendix A.3.

**Baselines:** We compare against prompting-based and training-based (RL and SFT) baselines (details in Appendix A.4).

## 4.2 Results and Analysis

### 4.2.1 Main Results

Table 1 shows our main experimental results on five standard benchmarks. We compare our Noisy-SQL-3B model (trained separately on the BIRD and Spider datasets) with prompting-based methods and training-based methods. The key findings are:

Table 1: Execution Accuracy (EX) comparison of Noisy-SQL and baselines on various Text-to-SQL benchmarks. **Bold** marks the best, underline the second best.

| Methods | Models | EX (%) | | | | |
|---|---|---|---|---|---|---|
| | | SPIDER-SYN | SPIDER-REAL | BIRD-DEV | SPIDER-DEV | SPIDER-TEST |
| **Prompting-based** | | | | | | |
| DAIL-SQLGao et al. (2023) | GPT-4 | - | - | 54.8 | 83.5 | 86.2 |
| MAC-SQLWang et al. (2024) | GPT-4 | - | - | 59.4 | 86.7 | 82.8 |
| DIN-SQLLiu et al. (2024a) | GPT-4o-mini | 64.3 | 62.0 | 50.7 | 70.3 | - |
| Solid-SQLLiu et al. (2024a) | GPT-4o-mini | 74.6 | 76.7 | 58.9 | 79.9 | - |
| **Training-based** | | | | | | |
| CodeS-7BLi et al. (2024a) | StarCoder-7B | 70.0 | 77.2 | 57.17 | 80.3 | - |
| Omni-SQL-7BLi et al. (2025) | Qwen2.5-Coder-7B | 69.7 | 76.2 | **63.9** | 81.2 | **87.9** |
| SENSE-13BYang et al. (2024) | CodeLlama-13B | 70.2 | 76.6 | 55.5 | <u>84.1</u> | 86.6 |
| Think2SQL-7BPapicchio et al. (2025) | Qwen2.5-Coder-7B | 78.0 | - | 56.1 | - | 82.4 |
| Noisy-SQL-3B (bird) | Qwen2.5-Coder-3B | <u>81.4</u> | <u>77.4</u> | <u>60.5</u> | 82.6 | 83.9 |
| Noisy-SQL-3B (spider) | Qwen2.5-Coder-3B | **83.6** | **79.9** | 54.3 | **84.7** | <u>86.7</u> |

***(1) Our method improves model robustness.*** Addressing robustness in Text-to-SQL is a core goal of this work. As Table 1 shows, when trained on the Spider dataset, our method achieves execution accuracies of 83.6% and 79.9% on the SPIDER-SYN and SPIDER-REAL benchmarks, respectively. These results outperform all baselines and set new state-of-the-art (SOTA) records.

***(2) Our method delivers strong overall performance with fewer parameters.*** Using only 3B parameters, Noisy-SQL consistently beats much larger LLMs. It outperforms GPT-4-based methods (DAIL-SQL, MAC-SQL) on BIRD-DEV and SPIDER-TEST, and surpasses all GPT-4o-mini baselines on all four datasets, including the robustness-focused Solid-SQL. Among training-based models, it achieves the highest accuracy (84.7%) on SPIDER-DEV, surpassing SENSE-13B and Omni-SQL-7B, while also attaining a competitive 60.5% on the challenging BIRD-DEV dataset.

***(3) Our method shows strong cross-domain generalization.*** When tested on the Spider domain using the Noisy-SQL-3B model trained only on the BIRD dataset—which has a completely different schema—it still performs well, reaching 82.6% and 83.9% execution accuracy on SPIDER-DEV and SPIDER-TEST, respectively. Notably, this cross-domain model ranks second best among all baselines on the SPIDER-SYN (81.4%) and SPIDER-REAL (77.4%) robustness benchmarks, demonstrating the strong generalizability of our method.

### 4.2.2 Ablation Study

We conduct an ablation study using Qwen2.5-Coder-3B to evaluate the individual contributions of Semantic Enhancement, and Noise-Aware Training. Results are shown in Table 2.

***(1) Effectiveness of Semantic Enhancement:*** Applying either Schema or Question Enhancement independently improves execution accuracy. However, Question Enhancement yields the largest gains, with absolute improvements of 3.3% and 3.95% on SPIDER-SYN and SPIDER-REAL, respectively, highlighting the critical need to resolve query ambiguity. Combining both strategies further improves

Table 2: Ablation Study. The ✓ indicates that the corresponding module is applied.

| Base Model | Semantic Enhancement | | Noise-Aware | EX (%) | | | | |
|---|---|---|---|---|---|---|---|---|
| | Schema | Question | | BIRD-DEV | SPIDER-SYN | SPIDER-REAL | SPIDER-DEV | SPIDER-TEST |
| Qwen2.5-Coder-3B | | | | 56.19 | 75.5 | 71.65 | 77.6 | 79.0 |
| | ✓ | | | 56.52 | 76.8 | 72.0 | 78.8 | 80.8 |
| | | ✓ | | 57.89 | 78.8 | 75.6 | 80.6 | 81.8 |
| | ✓ | ✓ | | 58.15 | 80.1 | 76.0 | 81.5 | 82.3 |
| | ✓ | ✓ | ✓ | **60.5** | **81.4** | **77.4** | **82.6** | **83.9** |

performance (e.g., reaching 80.1% on SPIDER-SYN), demonstrating their strong complementarity in addressing semantic poverty.

***(2) Crucial Role of Noise-Aware Training:*** Incorporating Noise-Aware GRPO training on top of semantic enhancements achieves the best overall performance. It significantly boosts accuracy on the challenging BIRD-DEV dataset by 2.35% (from 58.15% to 60.5%) and reaches 83.9% on SPIDER-TEST. This demonstrates that while semantic enhancement is beneficial, it inevitably introduces noise. The noise-aware mechanism effectively filters out misleading hints during CoT reasoning, maximizing the utility of synthesized semantics and ensuring robust SQL generation.

### 4.2.3 CASE STUDY ON NOISE-AWARE

To demonstrate the effectiveness of the noise-aware method under semantic noise, we compare the reasoning trajectories of the standard GRPO algorithm and our noise-aware algorithm in Figure 3.

Using the noise injection technique described in Section 3.2, we replaced the correct sub-question hint, "- `stealth power` means `WHERE power_name = 'Stealth'`", with a noisy hint, "- `stealth power` means `WHERE power_name = 'Super Strength'`".

Faced with this misleading hint, the model trained with standard GRPO is vulnerable. As shown by the red text in its reasoning, the model is distracted by the noise and incorrectly includes `WHERE T3.power_name = 'Super Strength'` in the final SQL query.




Figure 3: Impact of Noise-Aware

By contrast, with the noise-aware mechanism (highlighted in green), the model detects the noise during its Chain-of-Thought process, corrects the flawed hint back to `WHERE power_name = 'Stealth'`, and generates the correct SQL condition.

### 4.2.4 CASE STUDY ON SEMANTIC ENHANCEMENT

To clearly demonstrate the effectiveness of our semantic enhancement method, we compare the SQL generation results for a query with and without semantic enhancement in Figure 4. The natural language query is $Q$: *What are the names of the faculties whose birthplace is not 'Little Lever Urban District'?*. The main difficulty of this query is the use of the term *faculties* to mean teachers, an unusual expression causing semantic ambiguity that small-scale models find hard to handle. Moreover, the database schema $S$ contains only the `teacher` table and no `faculties` table, increasing the semantic gap.

As shown at the bottom-left of Figure 4, the baseline model without semantic enhancement fails to link `faculties` to the `teacher` table during inference. This

results in the model mistakenly assuming the existence of a `Faculty` table, producing an incorrect SQL query: `SELECT Name FROM Faculty WHERE ...`. In contrast, at the bottom-right of Figure 4, after applying our semantic enhancement with the SynGen model, the generated hints clearly indicate that *faculties* corresponds to the `teacher` table. Using this guidance, the model correctly outputs the SQL statement: `SELECT Name FROM teacher WHERE Hometown != ...`.

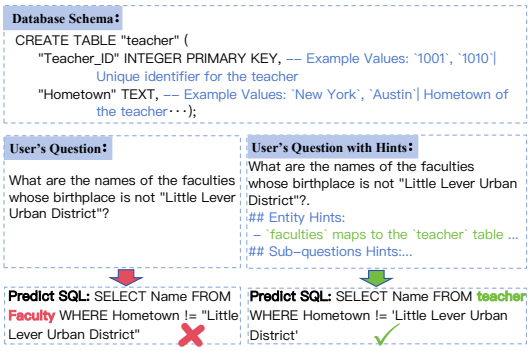



Figure 4: Impact of Semantic Enhancement

#### 4.2.5 Noise-Aware Capability Analysis

Because noise detection occurs implicitly within the model's Chain-of-Thought (CoT)Wei et al. (2022) reasoning, automatically calculating standard metrics like precision and recall is difficult. Instead, we evaluate the noise-aware capability of Noisy-SQL-3B by measuring its performance drop after injecting noise into five validation sets (detailed in Section 3.2). Figure 5 illustrates this absolute performance drop compared to the standard GRPO baseline, where a smaller drop indicates superior resilience against data corruption.

As shown in Figure 5, the standard GRPO algorithm degrades significantly when exposed to noisy hints, with accuracy dropping by 2.40% on *spider-real* and 1.80% on both *spider-syn* and *spider-dev*. In stark contrast, our noise-aware method maintains a much smaller performance drop across all datasets. Notably, on *spider-syn*, *spider-dev*, and *spider-test*, our degradation is limited to merely 0.60%—three times lower than the baseline. Even on the challenging *spider-real* dataset, our method halves the performance loss from 2.40% to 1.20%. These results prove that our noise-aware mechanism can effectively perceive and filter out erroneous semantic noise introduced during the enhancement process.

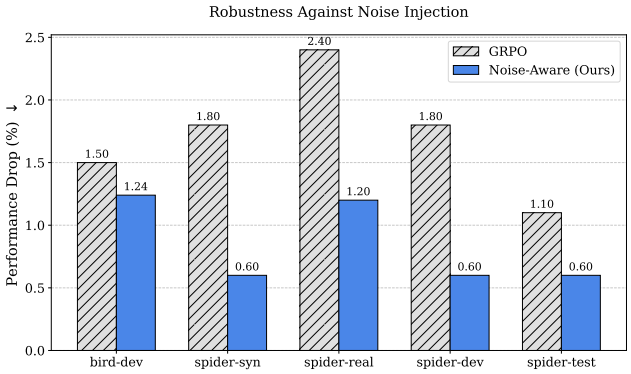



Figure 5: Performance degradation across different datasets after noise injection (lower drop indicates better robustness).

Table 3: Execution Accuracy (EX %) across query difficulty levels on BIRD-DEV.

| Semantic Enhancement | BIRD-DEV (EX %) | | | |
|---|---|---|---|---|
| | simple | moderate | challenging | total |
| w/o | 64.65 | 45.16 | 37.50 | 56.19 |
| w | **65.51** | **47.74** | **44.44** | **58.15** |

#### 4.2.6 Fine-Grained Analysis of Semantic Enhancement

We evaluate execution accuracy across query difficulty levels on the BIRD-DEV dataset to identify where semantic enhancement helps most. As shown in Table 3, performance consistently improves across all tiers. The impact is most pronounced on complex queries, yielding a 6.94% absolute increase (from 37.50% to 44.44%) on the `challenging` subset. This demonstrates that resolving semantic ambiguity is especially critical for difficult SQL generation tasks.

#### 4.2.7 EFFICIENCY ANALYSIS

To assess the practicality of our method, we analyzed inference efficiency. As shown in Table 4, for the same parameter size (Qwen-3B), although semantic enhancement adds extra input tokens, the lightweight model maintains fast inference. Its latency increases by only 0.3 seconds compared to the baseline GRPO, while improving execution accuracy on the Spider-syn dataset by 5.9%. Moreover, when compared to the larger Qwen-7B model with twice the parameters, our 3B model with the proposed method achieves 1.0% higher accuracy and reduces inference latency by 1.2 seconds. This shows our method offers clear benefits in both efficiency and accuracy, even against 7B-scale models.

Table 4: Efficiency and performance comparison on the Spider-syn dataset.

| Models | Methods | SPIDER-SYN (EX %) | Token Usage (tokens) | | | Latency (s) |
|---|---|---|---|---|---|---|
| | | | Input | Output | Total | |
| Qwen-3B | GRPO | 75.5 | 246.7 | 284.0 | 530.7 | 1.63 |
| Qwen-7B | GRPO | 80.4 | 246.7 | 307.1 | 553.8 | 3.13 |
| Qwen-3B | **Ours** | **81.4** | 622.5 | 315.2 | 937.7 | **1.93** |

#### 4.2.8 GENERALIZATION ANALYSIS

To evaluate generalizability, we tested our method on a different architecture (Llama-3.2-3B) and a smaller model (Qwen2.5-1.5B). As shown in Table 5, adding Semantic Enhancement consistently improves baseline performance. The improvements are particularly notable on robustness benchmarks, with absolute gains of 3.8% on SPIDER-SYN (Llama-3.2-3B) and 5.5% on SPIDER-REAL (Qwen2.5-1.5B). Incorporating the Noise-Aware module further increases accuracy on all datasets by effectively reducing enhancement noise. These consistent gains demonstrate that our full framework generalizes well across different model types and sizes. Additional analyses of complementarity with prior enrichment methods, cryptic schemas, the source of the gains, human validation, and multi-seed variance are reported in Appendix D.

Table 5: Execution Accuracy (EX %) demonstrating generalization across different models.

| Base Model | Semantic Enhancement | Noise-Aware | EX (%) | | | | |
|---|---|---|---|---|---|---|---|
| | | | BIRD-DEV | SPIDER-SYN | SPIDER-REAL | SPIDER-DEV | SPIDER-TEST |
| Llama-3.2-3B | | | 54.24 | 71.5 | 67.7 | 74.6 | 75.1 |
| | ✓ | | 55.54 | 75.3 | 71.3 | 77.1 | 77.8 |
| | ✓ | ✓ | **56.06** | **77.8** | **73.2** | **79.1** | **78.8** |
| Qwen2.5-1.5B | | | 55.87 | 75.9 | 68.7 | 77.5 | 78.0 |
| | ✓ | | 57.95 | 78.1 | 74.2 | 79.5 | 80.4 |
| | ✓ | ✓ | **59.19** | **79.3** | **76.6** | **81.0** | **82.0** |

## 5 LIMITATIONS

While our framework effectively reduces semantic ambiguity in Text-to-SQL tasks and addresses noise from semantic enhancement through noise-aware training, some limitations remain: ***(1) Difficulty in Identifying Hard Noise:*** Although the noise-aware mechanism detects some interference, it struggles to handle "hard noise," such as cases with very similar semantics or those requiring deep domain-specific understanding. A detailed analysis of these cases is given in Appendix C. ***(2) Reliance on Domain-Specific Training Data:*** During reverse semantic inference, the model's reasoning alone is insufficient to infer complex business logic or specialized knowledge. Currently, this knowledge depends heavily on the training data, which is costly to collect in real-world applications. Therefore, improving noise identification accuracy and reducing dependence on large domain-specific datasets are important areas for future work.

## 6 CONCLUSION

In this paper, we address the poor robustness of small-scale Text-to-SQL models caused by ambiguous database schemas and user queries. To bridge this semantic gap, we propose a Reverse Semantic Enhancement framework that trains a Semantic Synthesis Generator (SynGen) to dynamically enrich inputs using historical logs and database instances. Since automated enhancement inevitably introduces noise, we further design a Noise-Aware Training mechanism based on GRPO. This empowers our downstream model (Noisy-SQL) to proactively identify and filter out misleading hints during its Chain-of-Thought reasoning. Extensive evaluations demonstrate that, with only 3B parameters, Noisy-SQL achieves state-of-the-art accuracies on robustness benchmarks (81.4% on Spider-Syn, 77.4% on Spider-Real) and a competitive 60.5% on BIRD. Ultimately, this work provides a robust and computationally efficient paradigm for Text-to-SQL deployments in resource-constrained environments.

### REPRODUCIBILITY STATEMENT

Datasets, evaluation metrics, and implementation settings are reported in Section 4.1 and Appendix A, including data synthesis, SynGen training, noise injection, and Noisy-SQL training. Hyperparameter settings and reward weights are specified in Appendix A.3 and Appendix B. Experiments were run on 8 NVIDIA A100-SXM4-80GB GPUs, and training the 3B model took approximately 24 hours. To preserve double-blind anonymity, code and data are not released with this submission; we plan to release them publicly after the anonymity period.

### ETHICS STATEMENT

This work uses the public Spider and BIRD benchmarks and their robustness variants. It does not involve crowdsourcing or human subjects, so IRB approval is not applicable. The goal is to improve the robustness and efficiency of database querying. We did not identify significant direct negative societal impacts from this methodological improvement. Technical limitations, including hard noise and reliance on domain-specific training data, are discussed in Section 5 and Appendix C.

### AI USE STATEMENT

Large language models are a core component of the method. As described in Sections 3.1 and 3.2, Qwen2.5-72B-Instruct is used to synthesize semantically enhanced schema descriptions and question hints, and as a reward model that scores whether the reasoning trajectory identifies and corrects noisy hints. We take responsibility for the final content of this work, including artifacts produced with these models.

## REFERENCES

Moustafa Alzantot, Yash Sharma, Ahmed Elgohary, Bo-Jhang Ho, Mani Srivastava, and Kai-Wei Chang. Generating natural language adversarial examples. *arXiv preprint arXiv:1804.07998*, 2018.

Qifeng Cai, Hao Liang, Chang Xu, Tao Xie, Wentao Zhang, and Bin Cui. Text2sql-flow: A robust sql-aware data augmentation framework for text-to-sql. *arXiv preprint arXiv:2511.10192*, 2025.

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. Rsl-sql: Robust schema linking in text-to-sql generation. *arXiv preprint arXiv:2411.00073*, 2024.

Jipeng Cen, Jiaxin Liu, Zhixu Li, and Jingjing Wang. Sqlfixagent: Towards semantic-accurate sql generation via multi-agent collaboration. *arXiv preprint arXiv:2406.13408*, 2024.

Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. Dr. spider: A diagnostic evaluation benchmark towards text-to-sql robustness. *arXiv preprint arXiv:2301.08881*, 2023.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. Structure-grounded pretraining for text-to-sql. *arXiv preprint arXiv:2010.12773*, 2020.

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

Javid Ebrahimi, Anyi Rao, Daniel Lowd, and Dejing Dou. HotFlip: White-box adversarial examples for text classification. In Iryna Gurevych and Yusuke Miyao (eds.), *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pp. 31–36, Melbourne, Australia, July 2018. Association for Computational Linguistics. doi: 10.18653/v1/P18-2006. URL https://aclanthology.org/P18-2006/.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. Towards robustness of text-to-sql models against synonym substitution. *arXiv preprint arXiv:2106.01065*, 2021.

Chang Gao, Bowen Li, Wenxuan Zhang, Wai Lam, Binhua Li, Fei Huang, Luo Si, and Yongbin Li. Towards generalizable and robust text-to-sql parsing. *arXiv preprint arXiv:2210.12674*, 2022.

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

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yu Wu, YK Li, et al. Deepseek-coder: When the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Robin Jia and Percy Liang. Adversarial examples for evaluating reading comprehension systems. *arXiv preprint arXiv:1707.07328*, 2017.

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

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, et al. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, 33: 9459–9474, 2020.

Chaofan Li, Yingxia Shao, Yawen Li, and Zheng Liu. Sea-sql: Semantic-enhanced text-to-sql with adaptive refinement. *Frontiers of Computer Science*, 20(3):2003602, 2026.

Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. Codes: Towards building open-source language models for text-to-sql. *Proceedings of the ACM on Management of Data*, 2(3):1–28, 2024a.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, et al. Omnisql: Synthesizing high-quality text-to-sql data at scale. *arXiv preprint arXiv:2503.02240*, 2025.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 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, 2024b.

Geling Liu, Yunzhi Tan, Ruichao Zhong, Yuanzhen Xie, Lingchen Zhao, Qian Wang, Bo Hu, and Zang Li. Solid-sql: Enhanced schema-linking based in-context learning for robust text-to-sql. *arXiv preprint arXiv:2412.12522*, 2024a.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuxin Zhang, Ju Fan, Guoliang Li, Nan Tang, and Yuyu Luo. A survey of nl2sql with large language models: Where are we, and where are we going? *arXiv preprint arXiv:2408.05109*, 2024b.

Ilya Loshchilov and Frank Hutter. Sgdr: Stochastic gradient descent with warm restarts. *arXiv preprint arXiv:1608.03983*, 2016.

Multi-Linguality Multi-Functionality Multi-Granularity. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. *arXiv preprint arXiv:2402.03216*, 2024.

Simone Papicchio, Simone Rossi, Luca Cagliero, and Paolo Papotti. Think2sql: Reinforce llm reasoning capabilities for text2sql. *arXiv preprint arXiv:2504.15077*, 2025.

Xinyu Pi, Bing Wang, Yan Gao, Jiaqi Guo, Zhoujun Li, and Jian-Guang Lou. Towards robustness of text-to-sql models against natural and realistic adversarial table perturbation. *arXiv preprint arXiv:2212.09994*, 2022.

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

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. Route: Robust multitask tuning and collaboration for text-to-sql. *arXiv preprint arXiv:2412.10138*, 2024.

Shuhuai Ren, Yihe Deng, Kun He, and Wanxiang Che. Generating natural language adversarial examples through probability weighted word saliency. In *Proceedings of the 57th annual meeting of the association for computational linguistics*, pp. 1085–1097, 2019.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Hao Shen, Ran Shen, Gang Sun, Yiling Li, Yifan Wang, and Pengcheng Zhang. Sequential feature augmentation for robust text-to-sql. In *2023 International Conference on Algorithms, Computing and Data Processing (ACDP)*, pp. 217–223. IEEE, 2023.

Jinwang Song, Hongying Zan, Kunli Zhang, Lingling Mu, Yingjie Han, Haobo Hua, and Min Peng. Jolt-sql: Joint loss tuning of text-to-sql with confusion-aware noisy schema sampling. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 6051–6064, 2025.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, et al. Mac-sql: A multi-agent collaborative framework for text-to-sql. *arXiv preprint arXiv:2312.11242*, 2024.

Xuezhi Wang, Haohan Wang, and Diyi Yang. Measure and improve robustness in nlp models: A survey. In *Proceedings of the 2022 conference of the North American chapter of the association for computational linguistics: human language technologies*, pp. 4569–4586, 2022.

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

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

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887*, 2018.

Tianshu Zhang, Kun Qian, Siddhartha Sahai, Yuan Tian, Shaddy Garg, Huan Sun, and Yunyao Li. Evoschema: Towards text-to-sql robustness against schema evolution. *arXiv preprint arXiv:2603.10697*, 2026.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. A survey of large language models. *arXiv preprint arXiv:2303.18223*, 1(2):1–124, 2023.

# A Experimental Settings

## A.1 Datasets

This paper mainly utilizes the Spider(Yu et al., 2018) and Bird(Li et al., 2024b) datasets, which are the most widely used datasets for the Text-to-SQL task. The Spider dataset contains 8,659 training examples, 1,034 development examples, and 2,147 test examples. The Bird dataset is more complex than Spider and better reflects real-world scenarios, comprising 9,428 training examples and 1,534 development examples.

In addition, this paper also employs two variants of the Spider dataset: Spider-Syn(Gan et al., 2021) and Spider-Real(Deng et al., 2020), both derived from the Spider-dev dataset.

Spider-Syn replaces natural language queries with synonyms; for example, the query `"what is the type of the document named 'David CV'"` becomes `"what is the type of the file named 'David CV'"` in Spider-Syn. This approach breaks the direct correspondence between natural language queries and database fields to simulate attacks on schema linking, thereby challenging the model's robustness. Spider-Real not only replaces or removes column names but also substitutes phrases in natural language queries; for instance, "How many pets have a greater weight than 10" is changed to "How many pets are over 10 lbs?" Therefore, we select these two datasets to evaluate the robustness of our model.

## A.2 Evaluation Metrics

To evaluate the performance of our proposed method, we adopt **Execution Accuracy (EX)** as the primary evaluation metric. EX is a widely used and robust metric in Text-to-SQL tasks that measures the functional correctness of the generated SQL queries. Unlike exact set matching (EM) which compares the SQL query components, EX evaluates the queries by executing both the predicted SQL and the ground-truth (gold) SQL against the target database and comparing their output result sets. A prediction is deemed correct if and only if the execution results are completely identical.

Formally, given a dataset containing $N$ samples, let $\hat{y}_i$ and $y_i$ denote the predicted SQL and the gold SQL for the $i$-th query, respectively. Let $D_i$ represent the corresponding database instance, and $V(q, D)$ denote the returned result set of executing a SQL query $q$ on database $D$. The Execution Accuracy is calculated as follows:

$$\text{EX} = \frac{1}{N} \sum_{i=1}^{N} \mathbb{I}(V(\hat{y}_i, D_i) = V(y_i, D_i))$$

where $\mathbb{I}(\cdot)$ is the indicator function that returns 1 if the condition is true (i.e., the execution results match exactly) and 0 otherwise.

For the BIRD dataset, we apply the official execution accuracy evaluation script. For the SPIDER dataset and its robustness variants, we employ the `spider-test-suite` script, which provides a rigorous evaluation environment to accurately assess the semantic equivalence of the generated SQL queries and minimize false negatives.

### A.3 Implementation Details

Our implementation process consists of several stages. In this section, we detail the specific configurations and settings for each phase.

1. **Data Synthesis via Reverse Semantic Inference:** In this phase, we employ `Qwen2.5-72B-Instruct` as the semantic inference model to synthesize the semantically enhanced data required for training the SynGen model. We utilize LMDeploy[1] as the inference deployment framework. The sampling temperature is set to 1.0, while all other generation parameters follow the default settings of the OpenAI[2] Python library.
2. **SynGen Model Training:** This phase focuses on training the semantic synthesis generator, SynGen. We conduct the training utilizing the open-source repository DB-GPT-HUB[3]. The learning rate is set to $2 \times 10^{-5}$, with the learning rate scheduler configured to `cosine_with_restarts` Loshchilov & Hutter (2016). We set the batch size to 2 and the number of training epochs to 5. The `max_source_length` and `max_target_length` are constrained to 2048 and 512, respectively. Furthermore, DeepSpeed is configured with ZeRO Stage 2 for optimization.
3. **Noise Injection:** The primary objective of noise injection is to prepare the dataset for subsequent noise-aware training. We utilize the BGE-M3Multi-Granularity (2024) model to generate embeddings for similarity search. A similarity threshold of 0.9 is established; candidate hints with a similarity score exceeding this threshold are selected as semantic noise and used to replace the original correct hints.
4. **Noisy-SQL Model Training:** This phase involves training the noise-aware Text-to-SQL model. We apply reinforcement learning utilizing the VERL[4] framework. The key hyperparameters are configured as follows: `max_prompt_length` = 4096, `max_response_length` = 2048, `rollout_n` = 5, learning rate = $1 \times 10^{-6}$,`ppo_micro_batch_size_per_gpu` = 8, and `total_epochs` = 30. All other parameters are kept at their default values. The reward weights are assigned as follows: `Noise-Aware_Reward` = 1, `Execution_Reward` = 3 (comprising Syntax Reward = 1 and Accuracy Reward = 2), and `Format_Reward` = 1.

Our experiments primarily utilize `Qwen2.5-Coder-3B-Instruct` (Hui et al., 2024) as the base model. To demonstrate the generalizability of our approach, we also report results based on the `Qwen2.5-Coder-1.5B -Instruct` model. All experiments were conducted on a server equipped with 8 NVIDIA A100-SXM4-80GB GPUs. The training process for the 3B model took approximately 24 hours.

### A.4 Baselines

To comprehensively evaluate the effectiveness of our proposed framework, we compare our model against two categories of state-of-the-art baselines:

#### A.4.1 Prompting-based Methods

These methods utilize in-context learning and advanced prompting strategies with large-scale closed-source models (e.g., GPT-4o-mini and GPT-4):

- **DAIL-SQL** Gao et al. (2023): Employs efficient example selection and structured organization formats to leverage the high-quality generation capabilities of GPT-4.
- **MAC-SQL** Wang et al. (2024): Features a multi-agent framework comprising a decomposer agent for few-shot chain-of-thought reasoning, supported by auxiliary agents for sub-database retrieval and SQL refinement.

[1] `https://github.com/InternLM/lmdeploy`
[2] `https://github.com/openai/openai-python`
[3] `https://github.com/eosphoros-ai/DB-GPT-Hub`
[4] `https://github.com/volcengine/verl`

- **DIN-SQL** Pourreza & Rafiei (2024): Decomposes the Text-to-SQL task into four modular sub-problems, including schema linking, query classification, SQL generation, and self-correction.
- **Solid-SQL** Liu et al. (2024a): A robustness-oriented framework that enhances performance through improved schema linking and a multi-stage verification mechanism.

#### A.4.2 TRAINING-BASED METHODS

These methods involve fine-tuning or pre-training open-source models to achieve high domain-specific accuracy at a smaller parameter scale:

- **CodeS** Li et al. (2024a): A model series designed specifically for Text-to-SQL, utilizing incremental pre-training on SQL-specific corpora to address schema linking and domain adaptation challenges.
- **Omni-SQL** Li et al. (2025): A Qwen-based model that utilizes a multi-objective optimization approach and large-scale synthetic data to achieve high execution accuracy.
- **SENSE**Yang et al. (2024): A fine-tuned model focused on strengthening the model's perception of complex schema structures and cross-table relations.
- **Think2SQL** Papicchio et al. (2025): A reasoning-interleaved training framework that injects explicit logical reasoning chains before SQL generation to enhance overall robustness.

## B HYPERPARAMETER SENSITIVITY ANALYSIS

### B.1 REWARD FUNCTION WEIGHTS

Table 6: Hyperparameter sensitivity analysis of reward function weights on the BIRD-dev dataset.

| Reward Function Weights | | | | BIRD-DEV |
|---|---|---|---|---|
| `syntax` | `acc` | `format` | `noise_aware` | EX (%) |
| 1 | 1 | 1 | 0 | 58.15 |
| 1 | 1 | 1 | 1 | 60.12 |
| 1 | 2 | 1 | 1 | **60.50** |
| 1 | 1 | 1 | 2 | 60.32 |

To evaluate the sensitivity of reward function weights, we tested various reward function weight combinations on the BIRD-DEV dataset. As shown in Table 6, omitting the noise-aware reward (`noise_aware`=0) yields the lowest execution accuracy (58.15%). Introducing this reward (`noise_aware`=1) immediately boosts performance to 60.12%, confirming the critical role of the noise perception mechanism.

Furthermore, increasing the execution accuracy reward weight (`acc`=2) pushes the performance to a peak of 60.50%, highlighting its particular importance for Text-to-SQL tasks. Importantly, as long as the noise-aware reward is active, the execution accuracy consistently remains above 60.0% across different weight settings. This demonstrates that our framework is highly stable and robust to hyperparameter fluctuations.

### B.2 NOISE SIMILARITY THRESHOLD

We further analyzed the impact of the similarity threshold used during noise selection on the model's performance.

As shown in Figure 6, when the similarity threshold is set to 0.7 or 0.99, the execution accuracy drops to 59.91% and 59.39%, respectively. Evidently, when the semantic difference of the noise is too large (e.g., 0.7), it becomes too easy to identify, thereby weakening the model's ability to learn robust noise perception. Conversely, when the semantic similarity is too high (e.g., 0.99) and the noise closely resembles positive samples, the model may become confused. Therefore, a moderate

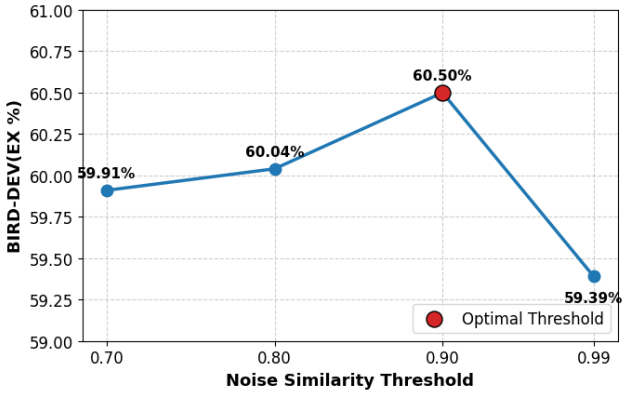



Figure 6: Impact of Noise Similarity Threshold on the BIRD-DEV dataset.

similarity threshold (e.g., 0.9) strikes the optimal balance, enabling the model to develop superior noise awareness capabilities.

## C Error Analysis

Although our proposed Noisy-SQL significantly improves robustness against semantic perturbations, a closer inspection of failure cases reveals several challenging scenarios where the model still struggles. By analyzing the reasoning trajectories of incorrect predictions, we identify two fundamental limitations of current noise-aware mechanisms: (1) knowledge-dependent semantic ambiguity, and (2) highly similar semantic overlap.

### C.1 Case Study 1: Knowledge-Dependent Semantic Ambiguity

In some cases, distinguishing noise requires implicit domain knowledge not explicitly provided in the schema or query. Without this context, the model cannot differentiate between a noisy hint and the correct interpretation.

Consider a query about a school with "Charter number 40". The generated hint is:

```
oversees the school with Charter number 40 means WHERE
CharterNum = '40'
```

Semantically, this mapping aligns perfectly with the query, leading the model to accept it as valid:

*<think>*
*...*
*- The condition WHERE CharterNum = '40' is correctly applied.*
*- No noise data is present, so we can proceed with the provided hints.*
*</think>*

However, the ground-truth SQL requires `CharterNum = '0040'` because the database stores charter numbers as zero-padded 4-digit strings. Although schema example values (e.g., `'0728'`, `'0811'`) weakly suggest this constraint, it is never explicitly defined.

Identifying this noise relies on external knowledge of specific data formats. To the model, both `'40'` and `'0040'` appear as valid semantic interpretations, leaving it no basis to reject the noise. This highlights that current mechanisms, which rely on detecting semantic inconsistencies, struggle when disambiguation depends on latent domain knowledge.

### C.2 Case Study 2: Highly Similar Semantic Overlap

Another challenging scenario arises when the generated noise is highly semantically similar to the ground-truth data, making it difficult for the model to detect.

Consider the following example. The generated hint is:

```
List the schools and the corresponding average writing
scores means SELECT sname, AvgScrWrite
```

In the schema, two columns are relevant and semantically adjacent:

- `satscores.sname`: annotated as "School name"
- `schools.School`: annotated as "Name of the school"

As seen here, both `sname` in the `satscores` table and `School` in the `schools` table represent the concept of a school name. The generated hint incorrectly suggests using `sname`. Despite this being a noisy hint, Noisy-SQL fails to recognize it as such.

As shown in the reasoning trajectory:

*<think>*
*...*
*- The correct columns to select are sname and AvgScrWrite.*
*- No noise data is present, so we can proceed with the provided hints.*
*</think>*

Consequently, Noisy-SQL mistakenly relies on this noisy hint to generate the final SQL statement, resulting in an error. This demonstrates that when noise data shares highly similar semantics with the target labels, it remains exceedingly difficult for Noisy-SQL to distinguish and filter out the interference.

# D Additional Analyses

This appendix reports analyses that were not in the main experiments. Numbers below are taken from those follow-up runs. The tables in the main paper continue to report the primary run, so a mean over seeds can differ slightly from the corresponding main-text entry.

## D.1 Complementarity with Prior Enrichment Methods

SEA-SQL and CodeS primarily retrieve cell values and native comments that are already stored in the database. When those comments are missing, as is common on Spider and parts of BIRD, they do not synthesize new schema descriptions or question-level hints. JOLT-SQL instead targets structural schema-linking errors, such as a linker dropping a required column. Our setting is different: reverse semantic inference synthesizes the missing comments and hints, and noise-aware training is aimed at semantic errors in those synthesized hints.

The same generated comments and hints can be inserted into other systems without retraining them. Figure 7 shows this plug-in comparison. On Spider, SEA-SQL rises from 77.3 to 80.3, CodeS-1B from 76.0 to 78.2, and a JOLT-format baseline from 64.6 to 77.3. On BIRD, the same three systems rise from 52.2 to 54.3, from 31.7 to 33.6, and from 19.1 to 38.5. The gains on the JOLT-format baseline are +12.7 on Spider and +19.4 on BIRD.

A schema with no native comment makes the difference concrete. For a `department` table whose `Creation` column stores years, an extractive prompt mainly appends example cell values. The synthesized prompt instead adds that `Creation` is the year the department was created, that "year" maps to this column, and that "most departments established" corresponds to an ordering clause.

## D.2 Cryptic Schemas

To isolate schema poverty, we evaluate on the SNAILS benchmark and report Least EX, the execution accuracy on the most heavily obfuscated schemas (Figure 8). Qwen2.5-Coder-3B-Instruct with semantic enhancement reaches 27.82, compared with 23.25 for GPT-3.5, 12.30 for CodeS-15B, and 10.38 for Phind-CodeLlama-34B-v2. Removing semantic enhancement drops our result from 27.82 to 22.87.

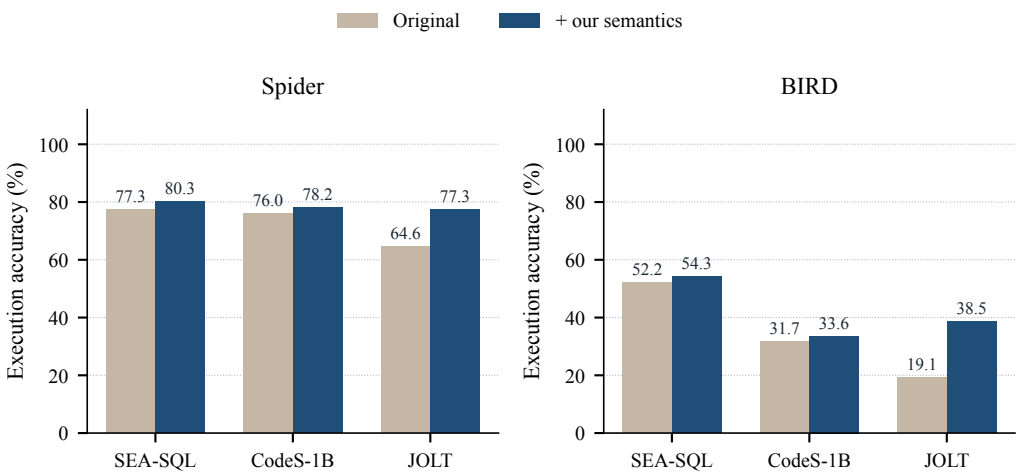



Figure 7: Execution accuracy before and after inserting our generated schema comments and question hints into SEA-SQL, CodeS-1B, and a JOLT-format baseline.

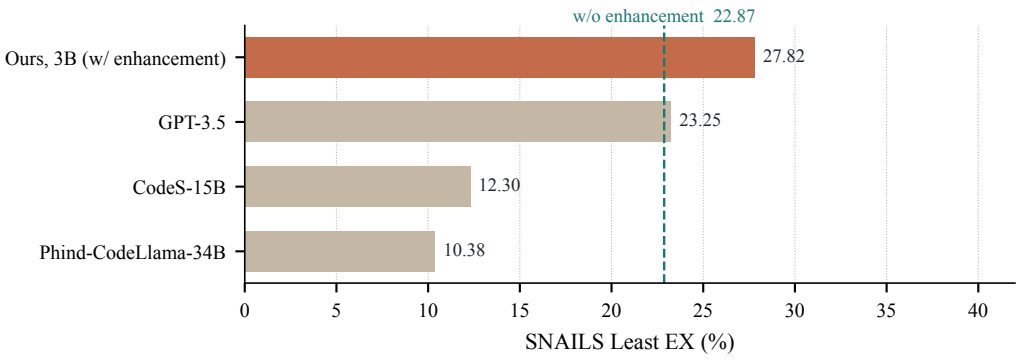



Figure 8: SNAILS Least EX. The dashed line is our 3B model without semantic enhancement.

### D.3 Source of the Gain

Figure 9 separates three comparisons on BIRD-dev, all using Noisy-SQL-3B unless noted.

The left panel compares semantic-enhancement sources. Using the untuned 3B model as the source yields 53.91 EX. Replacing it with SynGen-3B yields 60.50, and using Qwen2.5-72B-Instruct directly as the enhancer yields 62.11. The distilled 3B generator therefore recovers most of the teacher, with a remaining gap of 1.61.

The middle panel keeps the 72B model but removes reverse semantic inference: the teacher is prompted with only the schema and the question. That direct prompt reaches 57.91, against 60.50 for SynGen, a gap of 2.59.

The right panel changes only the synthesis teacher. The 3B model without a teacher scores 56.19. A Qwen3.6-35B-A3B teacher reaches 58.37, and the 72B teacher reaches 60.50. A smaller teacher still helps; the 72B teacher gives the larger gain.

The 72B judge is used only while training. Each GRPO step samples 5 outputs for a batch of 512 questions, so the judge scores 2,560 responses per step. On 8 A100-80GB GPUs the full training run takes about 24 hours. After training, the judge is discarded, and inference uses the 3B model alone, with the same cost as standard GRPO. That 3B model fits on a single consumer GPU such as an RTX 4090.

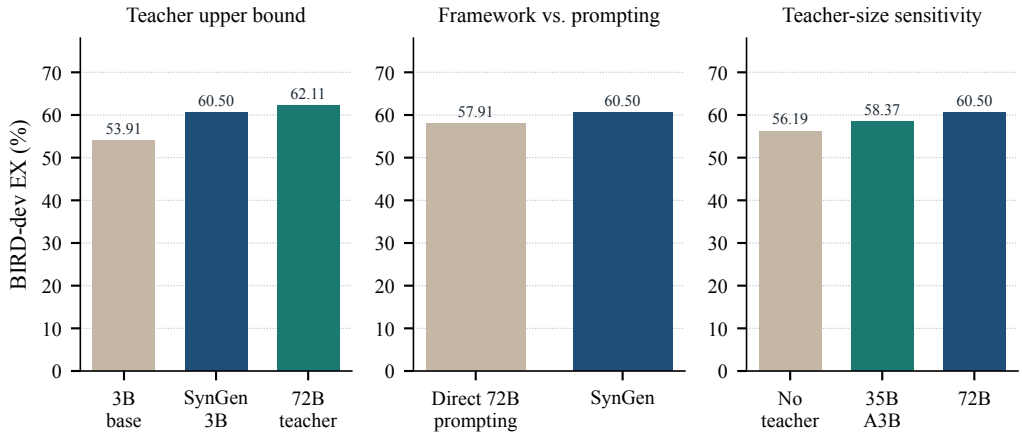



Figure 9: BIRD-dev execution accuracy under three separate comparisons: the 72B teacher as an upper bound on SynGen, direct 72B prompting without reverse inference, and synthesis teachers of different sizes. The 53.91 bar is the untuned 3B enhancer, not the GRPO ablation baseline of 56.19 reported in the main paper.

### D.4 Human Checks of Semantics and of the Judge

We manually graded 200 semantically enhanced BIRD examples on a three-point scale: 3 completely correct, 2 partially incorrect, and 1 completely wrong (Figure 10, left). Schema comments average 2.895, with 179 correct, 16 partial, and 5 wrong (89.5% perfect). Question hints average 2.755, with 169 correct, 13 partial, and 18 wrong (84.5% perfect). The partial and wrong cases are the noise that noise-aware training is meant to filter.

We also checked the 72B noise judge on 200 instances, 150 with injected noise and 50 clean. Its precision is 100.0%, recall is 84.0% (it misses 24 of the 150 noisy instances), and F1 is 91.3%. Against human scores on the same rubric (1.0 fully corrected, 0.7 partially corrected with no false positives, 0.4 partially corrected with false positives, and $-0.5$ failed to identify noise), exact agreement is 84.5%. The mean score difference, LLM minus human, is $+0.1335$: the judge is slightly more lenient, mostly on the 22 cases where a human assigned $-0.5$ and the judge assigned 1.0 (Figure 10, right).

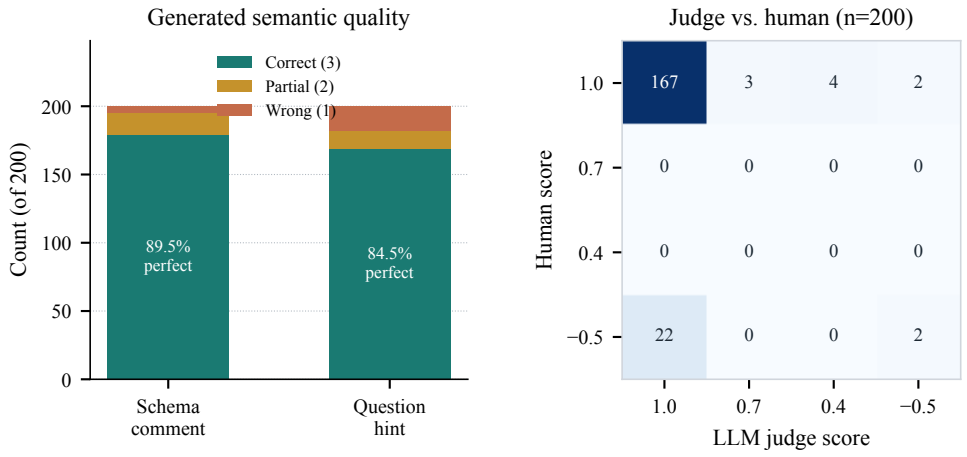



Figure 10: Left: manual grades of 200 generated schema comments and 200 question hints. Right: counts of human score versus 72B-judge score on 200 noise-judgment instances.

### D.5 Reproduced Robustness Baselines

Several robustness numbers were missing from the original baseline papers. We reproduced MAC-SQL and DAIL-SQL with GPT-4o, and Think2SQL-7B on Spider-Real (Figure 11). On Spider-Syn the execution accuracies are 64.1, 73.8, 78.0, and 83.6 for DAIL-SQL, MAC-SQL, Think2SQL-7B, and Noisy-SQL-3B. On Spider-Real they are 74.6, 80.1, 77.1, and 79.9. The Think2SQL Spider-Syn number is the published result; the other three baseline entries were reproduced by us.

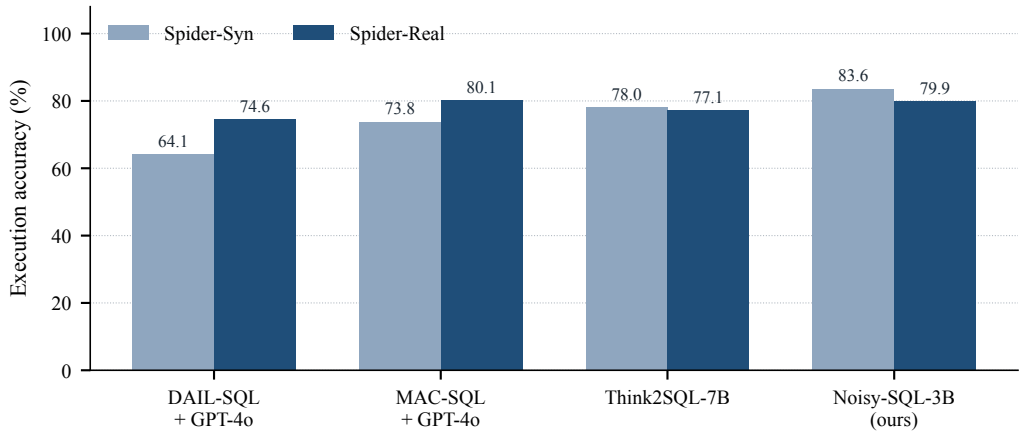



Figure 11: Robustness baselines completed on Spider-Syn and Spider-Real. MAC-SQL, DAIL-SQL, and Think2SQL on Spider-Real are our reproductions.

### D.6 Multi-Seed Variance

We repeated the main BIRD-trained and Spider-trained runs, and the Qwen2.5-Coder-3B ablation, with three random seeds. Figure 12 plots the ablation endpoints. Table 7 and Table 8 list mean $\pm$ standard deviation. Standard deviations fall between 0.10 and 1.04. On the BIRD-trained ablation, the full pipeline is 3.92 above the no-enhancement mean on BIRD-dev and 5.37 above it on Spider-Syn, in both cases larger than the seed-to-seed spread.

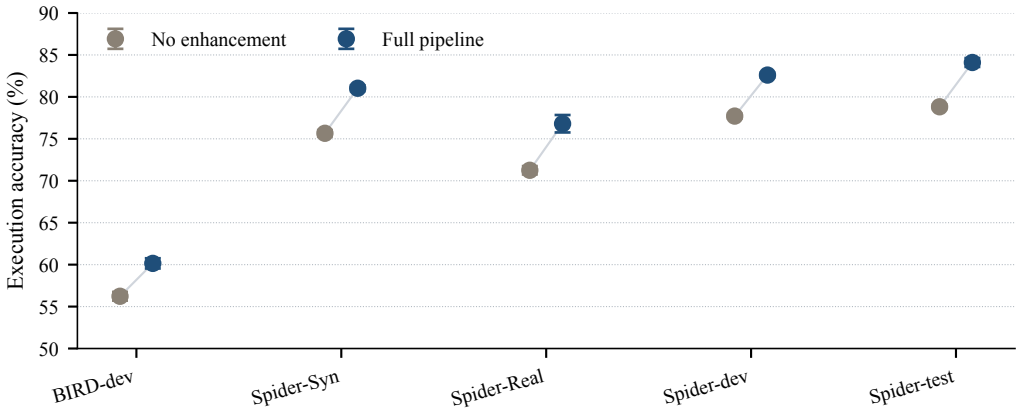



Figure 12: Three-seed means for Qwen2.5-Coder-3B with no semantic enhancement and with the full pipeline. Error bars are one standard deviation. The vertical axis starts at 50 so the intervals remain visible.

Table 7: Main results over three seeds (mean ± std, EX %). These means are not substituted for the primary-run numbers in Table 1.

| Method | Spider-Syn | Spider-Real | BIRD-dev | Spider-dev | Spider-test |
|---|---|---|---|---|---|
| Noisy-SQL-3B (BIRD) | 81.03 ± 0.35 | 76.80 ± 1.04 | 60.15 ± 0.61 | 82.60 ± 0.30 | 84.10 ± 0.53 |
| Noisy-SQL-3B (Spider) | 83.50 ± 0.10 | 79.17 ± 0.64 | 55.06 ± 0.82 | 83.90 ± 0.70 | 86.13 ± 0.42 |

Table 8: Ablation over three seeds on Qwen2.5-Coder-3B (mean ± std, EX %). A check mark denotes that the module is used.

| Schema | Question | Noise-Aware | BIRD-dev | Spider-Syn | Spider-Real | Spider-dev | Spider-test |
|---|---|---|---|---|---|---|---|
| | | | 56.23 ± 0.52 | 75.66 ± 0.38 | 71.25 ± 0.50 | 77.71 ± 0.30 | 78.82 ± 0.29 |
| ✓ | | | 56.70 ± 0.45 | 76.93 ± 0.40 | 72.42 ± 0.43 | 79.01 ± 0.28 | 80.55 ± 0.34 |
| | ✓ | | 57.69 ± 0.57 | 78.30 ± 0.42 | 75.43 ± 0.57 | 80.31 ± 0.34 | 81.67 ± 0.35 |
| ✓ | ✓ | | 58.35 ± 0.54 | 79.62 ± 0.38 | 76.35 ± 0.52 | 81.33 ± 0.34 | 82.56 ± 0.33 |
| ✓ | ✓ | ✓ | 60.15 ± 0.61 | 81.03 ± 0.35 | 76.80 ± 1.04 | 82.60 ± 0.30 | 84.10 ± 0.53 |