## A Notations and Symbols

The notations and symbols adopted in this paper are summarized in Table 1

**Table 1: Notations**

| Notation | Explanation |
|---|---|
| | *Common Variables* |
| $L$ | The Text-to-SQL translation function or model that maps a natural language question $Q$ to an SQL query $S$ (see Eq 1) |
| $D$ | Database, including its schema information |
| $\theta$ | Model parameters (see Eq 6) |
| $d$ | Embedding dimension (see Eq 22) |
| $b$ | Training batch size (see Eq 22) |
| | *Schema-related Variables* |
| $T$ | The set of tables in the database (see Eq 17) |
| $C$ | The set of columns in the database (see Eq 18) |
| $S_k$ | The relevant schemas (tables and columns) predicted by the SchemaLinker model (see Eq 33) |
| $P_s$ | The predicted set of schema links (see Eq 11) |
| $T_s$ | The ground truth set of schema links (see Eq 11) |
| | *SchemaLinker-related Variables* |
| $M_{teacher}$ | The teacher model (GPT-4o) used to generate CoT data (see Eq 3) |
| $CoT$ | Chain-of-Thought information (see Eq 3) |
| $\hat{y}_i$ | The label predicted by the teacher model (see Eq 3) |
| $\mathcal{T}$ | The set of all tasks in the multi-task learning framework (see Eq 7) |
| $I_{i(n)}$ | The instruction for the specific task $i$ corresponding to instance $n$ (see Eq 5) |
| $m_{i(n)}$ | The priority weight for task $i$ corresponding to instance $n$ (see Eq 7) |
| $N_i$ | The number of samples in task $i$ (see Eq 8) |
| $p_n$ | The probability of a single task instance $n$ being sampled (see Eq 7) |
| $E[samples_i]$ | The expected number of samples per task $i$ (see Eq 8) |
| $G$ | The size of the candidate set generated by the GRPO policy model |
| $\pi_\theta$ | Current policy model (see Eq 9) |
| $\pi_{old}$ | The old policy used to generate the candidate set (see Eq 9) |
| $A_j$ | The group-relative advantage for output $j$ (see Eq 9) |
| $r_i^{ratio}$ | The probability ratio between $\pi_\theta$ and $\pi_{old}$ (see Eq 9) |
| $\epsilon$ | Clipping parameter in GRPO (see Eq 9) |
| $\beta$ | KL divergence penalty coefficient in GRPO (see Eq 9) |
| $R_k$ | The initial reward based on format correctness (see Eq 10) |
| $R$ | The detailed reward function for SchemaLinker (see Eq 11) |
| $r_c, r_w, r_m, w_f$ | Weights for TP, FP, FN, and F1-score in the reward function $R$ (see Eq 11) |
| | *SAR-related Variables* |
| $c_i, t_i$ | Initial embeddings for columns and tables (see Eq 12, 13) |
| $E_q$ | The embedding representation of the question $Q$ (see Eq 14) |
| $E_s$ | The embedding representation of the ground-truth SQL query $S$ (see Eq 15) |
| $T_i^C$ | The "column-aware" table embedding, fused with its column information (see Eq 19) |
| $\hat{S}$ | The Schema-Aware embedding generated by the SAR model (see Eq 20) |
| $X^{(0)}, X^{(l)}$ | Input sequence and intermediate layer representations for the Transformer in contrastive learning (see Eq 22, 24) |
| $E_{final}$ | The final enhanced embedding after the Transformer and LayerNorm (see Eq 25) |
| $E_{\text{main}}$ | The anchor (main) embedding in contrastive learning (see Eq 26) |
| $E_{\text{similar}}$ | The embedding of a known similar sample corresponding to the anchor (see Eq 27) |
| $z_i, z_i^+$ | Anchor and its corresponding positive sample pair used in the contrastive loss (see Eq 28) |
| $\tau$ | Temperature coefficient in the contrastive loss (see Eq 28) |
| $M$ | Causal attention mask in the Transformer encoder (see Eq 23, 24) |
| $L_{\text{similarity}}$ | Similarity loss (see Eq 29) |
| $w_{\text{sim}}$ | Weight coefficient for the similarity loss (see Eq 30) |
| $L_{\text{total}}$ | The total loss for SAR training (see Eq 30) |
| | *POSG-related Variables* |
| $S_{ex}$ | The **Executability** score of the SQL query (see Eq 32) |
| $S_{sl}$ | The **Schema linking conformity** between the SQL query and the SchemaLinker output (see Eq 33) |
| $S_{ec}$ | The **Example consistency** of the SQL query with the retrieved K examples (see Eq 34) |
| $AST(S)$ | The Abstract Syntax Tree (AST) of the SQL query $S$ (see Eq 34) |
| $d(\cdot,\cdot)$ | The tree edit distance between two ASTs (see Eq 34) |
| $E$ | The set of valid candidate SQL queries after $S_{ex}$ filtering (see Eq 35) |
| $P$ | The Pareto-optimal set among the candidate queries (see Eq 35) |

## B Performance Evaluation of the Schema Linking Module

This section presents a comprehensive performance evaluation of our proposed **SchemaLinker**, assessing it from the dual perspectives of task effectiveness and computational efficiency during inference. To this end, we first introduce the implementation details and evaluation metrics. Subsequently, we conduct an in-depth analysis of the model's linking accuracy on the Spider and BIRD datasets across a progression of methods. Finally, by comparing the inference token consumption of different models, we further validate that our ultimate SchemaLinker method achieves both conciseness and high efficiency while maintaining superior performance.

### B.1 Implementation Details

The entire training pipeline was conducted on a computational setup consisting of two NVIDIA A800 (80GB) GPUs. We employed BF-16 mixed-precision training, facilitated by the `Accelerate` framework. The model was optimized using the AdamW optimizer with the following hyperparameter configuration:

- $\beta_1 = 0.9$
- $\beta_2 = 0.99$
- $\epsilon = 10^{-8}$
- Learning Rate = $5 \times 10^{-5}$

### B.2 Evaluation Metrics

For performance assessment, we evaluate table and column predictions using the metrics detailed in Table 2.

**Table 2: Core Evaluation Metrics for Schema Linking**

| Metric | Formula | Description |
|---|---|---|
| F1 Score | $\frac{2 \cdot TP}{2 \cdot TP+FP+FN}$ | The F1 score is the primary metric for evaluating table selection performance. As the harmonic mean of Precision and Recall, it provides a single, balanced score reflecting both the correctness and completeness of the identified tables. |
| Recall | $\frac{TP}{TP+FN}$ | Used to assess the completeness of schema element selection. It measures the model's ability to identify all ground-truth tables and columns required by the query, indicating how effectively it avoids missing necessary elements. |
| Matthews Correlation Coefficient (MCC) | $\frac{TP \times TN - FP \times FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}$ | Used specifically for evaluating column selection. Due to the significant data imbalance (a query uses only a small subset of all available columns), MCC provides a more reliable and balanced measure of the model's classification performance. |

- **Tables**: Performance on table selection is measured using the **F1 score**.
- **Columns**: Due to the significant data imbalance (i.e., SQL queries typically use only a small subset of available columns), we evaluate column selection performance using the **Matthews Correlation Coefficient**(MCC), as it provides a more reliable measure for imbalanced classification tasks.

### B.3 Results and Analysis

To validate the efficacy of the training methodologies of this study, we conducted evaluations on the development sets of the Spider and BIRD datasets subsequent to each training iteration. The results, as shown in Figure 1, demonstrate a consistent and progressive enhancement in the schema linking capabilities of the model at both the table and column levels across the evolution of the proposed methods—from Supervised Fine-Tuning and Distillation with Chain-of-Thought (CoT), to the integration of multi-task learning in Distillation CoT_MTL, and culminating in the Reinforcement Learning-based SchemaLinker.

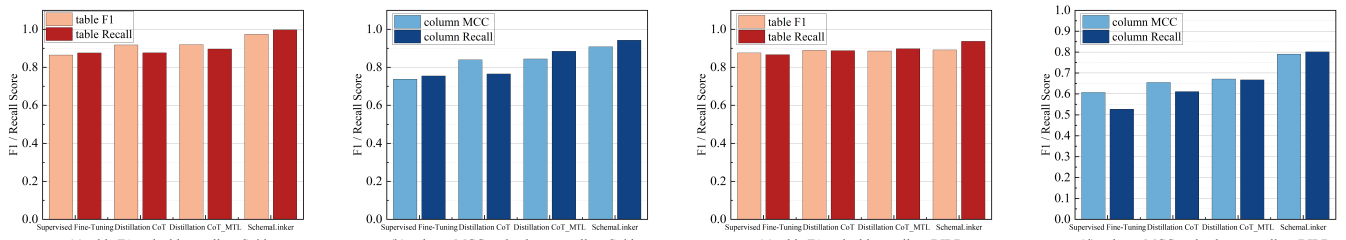
Specifically, on the Spider dataset, the SchemaLinker method achieved competitive performance, attaining a table-level F1-score of 97.38% and Recall of 99.69%, and a column-level MCC of 90.81% and Recall of 94.29%.

Specifically, on the Spider dataset, the SchemaLinker method achieved competitive performance, attaining a table-level F1-score of 97.38% and a recall of 99.69%. For column-level metrics, it achieved an MCC of 90.81% and a recall of 94.29%. These results represent substantial improvements of approximately 10.9%, 12.1%, 17.1%, and 18.8% respectively, over the baseline supervised fine-tuning approach.

A similar trend was observed on the more challenging BIRD development set. Although the overall performance metrics were marginally lower due to the inherent complexity of the dataset, the **SchemaLinker** method continued to demonstrate superior performance. This evidence strongly suggests that leveraging the capabilities of large models in conjunction with knowledge distillation and reinforcement learning strategies is a crucial pathway for augmenting the generalization and semantic understanding of a model regarding schema elements, thereby significantly elevating schema linking performance.

To further evaluate the computational efficiency of the SchemaLinker model, we compared its token consumption during the inference process, as presented in Table 3. The experimental results clearly demonstrate that SchemaLinker exhibits significant efficiency in both input and output token usage. Specifically, SchemaLinker achieves an average output token count of 175.7 ± 36.9, which is substantially lower than that of the base Qwen model (368 ± 96.2) and the Qwen model fine-tuned with Chain-of-Thought and multi-task learning (CoT_MTL, 205.6 ± 38.6).

This efficiency stems from SchemaLinker's multi-stage optimization process, which refines the model's verbosity without sacrificing its reasoning capabilities. The model is first taught a detailed reasoning process through knowledge distillation with CoT data. Subsequently, the final reinforcement learning stage, using GRPO, optimizes the model based on a precise reward function. This incentivizes the model to produce the most accurate and concise schema linking results, effectively pruning the verbose intermediate steps learned during the CoT phase. This process ensures that while the model internalizes a complex reasoning framework, its final output is distilled down to the essential, structured information required for the downstream task, thereby significantly reducing token consumption.



(a) table F1 and table recall on Spider (b) column MCC and column recall on Spider (c) table F1 and table recall on BIRD (d) column MCC and column recall on BIRD

**Figure 1: The four models (from left to right): supervised fine-tuning; knowledge distillation with CoT; multi-task CoT distillation; and distillation with reinforcement learning.**

**Table 3: Comparison of average Token Consumption on 300 samples from the BIRD dataset.**

| Model | Input Token | Output Token |
|---|---|---|
| Qwen | | 368.0 ± 96.2 |
| Qwen-Fintune(without CoT) | | 62.3 ± 22.6 |
| Qwen-Fintune(CoT) | 1886.9 ± 423.6 | 165.6 ± 32.4 |
| Qwen-Fintune(CoT_MTL) | | 205.6 ± 38.6 |
| SchemaLinker | | 175.7 ± 36.9 |

## C Dataset Details

We selected five widely adopted Text-to-SQL benchmark datasets: Spider, Spider 2.0 snow , BIRD, Beaver, and UniSQL. As shown in Table 4, Spider and its updated version, Spider 2.0 snow, are large-scale cross-domain datasets with a focus on complex database schemas and multi-turn queries, while BIRD emphasizes complex SQL queries in big data environments, and Beaver aligns with real-world enterprise data interactions featuring intricate query patterns. UniSQL, the fifth dataset, is constructed via stratified sampling across Spider, Spider 2.0 snow, BIRD, and Beaver, carefully balanced to ensure diversity in query and database complexity, providing robust generalization across a wide range of SQL query types and database structures. To simulate real-world scenarios where Retrieval-Augmented Generation (RAG) is used to enhance model performance, we extended the official development sets of these datasets.

In our implementation, we employed the GPT-4o model to generate example "Question–SQL–Schema" triples with structural similarity for each original triple across the validation or test sets of all target datasets, including Spider, BIRD, Spider 2.0 snow, and Beaver. To ensure the consistency and quality of these generated pairs, we implemented a rigorous automated verification pipeline. This process involved two critical checks: first, a custom script verified the **structural consistency** between each new SQL query and its original counterpart; second, we utilized Claude 4 Sonnet to perform an automated **semantic matching** check, confirming that the generated natural language question accurately corresponded to its SQL query. Only the triples that successfully passed both the structural and semantic verifications were retained. The resulting curated datasets are denoted as RAG_Spider, RAG_BIRD, RAG_Spider2.0, and RAG_Beaver, respectively. These RAG-augmented datasets not only serve as evaluation benchmarks for assessing the performance of the Schema-Augmented Retriever (SAR) but also provide high-quality supervised signals enriched with explicit retrieval cues to facilitate model learning.

As shown in Table 4, we collected a large number of Question–Schema–SQL triples from various public software development communities, data science forums, and code hosting platforms. To ensure data quality, we removed erroneous pairs where the SQL statements failed to execute successfully, as well as pairs with overly simplistic formats. The resulting cleaned and curated data was then structured into Question–Schema–SQL triples, which were used to train the Schema-aware module.

The training data for the Contrastive Learning module was generated based on the original training sets of BIRD and Spider, using the same generation methodology as that employed for the RAG datasets.

**Table 4: Comparison of dataset properties and RAG training examples**

| Feature | Spider | BIRD | Spider 2.0 snow | Beaver | UniSQL |
|---|---|---|---|---|---|
| Training Set | 8,659 | 9,428 | – | – | 8,659/9,428 |
| Development Set | 1,034 | 1,534 | – | – | – |
| Test Set | 1,534 | 2,147 | 547 | 209 | 1,024 |
| RAG Set | 3,102 | 4,176 | 1641 | 627 | 3,072 |
| Number of Databases | 200 | 95 | 158 | 6 | – |
| Domain Coverage | 138 | 37 | - | 465 | Mixed |
| Schema-aware | 31,171 | 31,171 | 31,171 | 31,171 | 31,171 |
| Contrastive Learning | 25,977 | 28,284 | – | – | – |

## D Evaluation Metrics Details

$$\text{VES} = \frac{\text{Execution Time}_{\text{ground-truth}}}{\text{Execution Time}_{\text{predicted}}} \tag{1}$$

The valid efficiency score (VES) is designed to evaluate the performance of correctly predicted queries. Unlike execution accuracy (EX), where a correct query receives a binary score of 1, VES provides a more nuanced measure of efficiency. It is calculated as the ratio of the execution time of the ground-truth query to the execution time of the predicted query, as shown in Equation 1.

Consequently, if the execution times of the predicted and ground-truth queries are identical, the VES score equals 1. If the predicted query executes faster than the ground-truth query, the VES score will be greater than 1. In practice, to ensure stable results, each correct predicted query and its corresponding ground-truth query are executed 100 times to record an average runtime.

Despite this averaging process, VES remains highly sensitive to variations in hardware, software, and system state, which is why EX is considered the primary and more dependable metric for the BIRD benchmark.

## E Model Details

The Large Language Models(LLMs) employed in this study, along with the comparative baseline models, are detailed in Table 5.

## F Detailed Experimental Results

This section provides the complete and detailed experimental results, which are presented in a summarized form in the main body of the paper.

Table 6 presents a comprehensive breakdown of model performance on the Spider dataset. It evaluates EX and EM across four difficulty levels (Easy, Medium, Hard, Extra). Table 7 details the performance on the more challenging BIRD dataset. It measures EX and the VES across three complexity levels (SIM, MOD, CHALL).

In both tables, results are shown for each framework (e.g., TA-SQL, SchemaRAG) when paired with different backbone LLMs. Values highlighted in bold or marked with a • indicate the top-performing model in each category.

## G Detailed Ablation Study

This section presents the ablation studies conducted to validate the contribution of the individual components of the SchemaRAG framework. We analyze the performance degradation when the SchemaLinker, SAR and POSG modules are individually removed.

Table 8 shows the results of this study on the Spider dataset. It compares the full SchemaRAG model against variants without SchemaLinker (w/o SchemaLinker), without SAR (w/o SAR) and without POSG(w/o POSG), demonstrating the impact of each component on EX and EM.

Table 9 provides the corresponding ablation results on the BIRD dataset. The same model variants are tested to show the importance of each component in this more challenging environment. Bolded values indicate the highest score among the three variants (full and ablated) for EX and VES.

## H SchemaRAG Prompt

This section provides a detailed explanation of the prompts used at each core stage of our SchemaRAG framework.

**Table 5: Descriptions of all the involved models**

| Model | Description |
|---|---|
| | *large language model* |
| CodeS*(PACMMOD 2024)* | A text-to-sql model based on StarCoder, developed using incremental pre-training techniques. |
| XiYanSQL*(Alibaba 2025)* | A text-to-sql model based on QwenCoder, trained using both fine-tuning and GRPO. |
| Qwen 2.5*(Alibaba 2025)* | An open-source, general-purpose LLM from the Tongyi Qianwen series, developed by Alibaba Cloud. |
| Llama 3*(Meta AI 2024)* | An open-source, general-purpose LLM from the Llama series, developed by Meta AI. |
| GPT-4o-mini*(OpenAI 2024)* | A lightweight, distilled LLM from the GPT-4o series, developed by OpenAI. |
| GPT-4o*(OpenAI 2024)* | A proprietary, general-purpose LLM from the GPT series, developed by OpenAI. |
| DeepSeek-V3*(DeepSeek 2024)* | A general-purpose LLM from the DeepSeek series, based on the MoE architecture, developed by DeepSeek. |
| GLM-4-Plus*(Zhipu AI 2024)* | A proprietary, general-purpose LLM from the GLM series, developed by Zhipu AI. |
| | *TEXT-to-SQL method* |
| TA-SQL*(ACL 2024)* | A Text-to-SQL method that mitigates hallucination in large language models through a task alignment strategy. |
| Din-SQL*(NeurIPS 2023)* | A Text-to-SQL method that addresses complex problems by employing a divide-and-conquer strategy. |
| ACT-SQL*(EMNLP 2023)* | A Text-to-SQL method that enhances the reasoning of large models via automatically generated Chain-of-Thought. |
| C3-SQL*(arXiv 2023)* | A Text-to-SQL method that improves zero-shot generation quality through prompt engineering and calibration. |

## H.1 PromptSchema

The schema used by our study is organized as shown in Figure 2.

## H.2 SchemaLinker

In the first stage of fine-tuning, both the prompt-CoT-response pairs for training and the final output are formatted as shown in Figure 3. Specifically, the CoT rationale follows a fixed template enclosed by `<think> ... </think>`, organized into three concise parts:

```
<think>
1. Understand key concepts:
   - Map natural-language phrases to schema elements:
     * 'heads of the departments' -> head_ID in head table
     * 'older than 56'          -> head.age

2. Analyze table relationships:
   - List relevant tables and explicit join paths:
     * management.head_ID - head.head_ID
     * management.department_ID - department.Department_ID

3. Identify key field(s) for filtering/aggregation:
   - Specify exact column(s) and justify:
     * key field for filtering: head.age
</think>
```

**Table 6: Comparison of EX and EM on the Spider dataset, ● indicates a result that is higher than the corresponding value of the SchemaRAG**

| Model | LLM | Metric | Easy | Medium | Hard | Extra | All |
|---|---|---|---|---|---|---|---|
| TA-SQL | GPT-4o-mini | EX | 93.5● | 90.8● | 77.6 | 64.5 | 85.0 |
| | | EM | 87.9● | 68.8● | 63.2 | 45.2● | 68.7● |
| | Deepseek-v3 | EX | 91.9● | 89.0 | 75.9 | 71.1 | 84.6 |
| | | EM | 85.5 | 81.4 | 63.2 | 59.6● | 75.8● |
| | Qwen-7B | EX | 90.3 | 88.5● | 54.8 | 45.9 | 76.4 |
| | | EM | 31.0 | 21.0 | 16.7 | 2.4 | 19.7 |
| | XiYanSQL-14B | EX | 91.4 | 85.2 | 59.3 | 49.7 | 76.6 |
| | | EM | 40.7 | 38.4 | 25.5 | 18.8 | 33.6 |
| | Llama-8B | EX | 86.3 | 78.5 | 61.8 | 43.6 | 72.0 |
| | | EM | 18.8 | 12.9 | 14.6 | 0.6 | 12.6 |
| DIN-SQL | GPT-4o-mini | EX | 91.1 | 79.8 | 64.9 | 43.4 | 74.2 |
| | | EM | 82.7 | 65.5 | 42.0 | 30.7 | 60.1 |
| | Deepseek-v3 | EX | 89.5 | 88.3 | 69.0 | 59.0 | 80.6 |
| | | EM | 76.2 | 42.2 | 33.3 | 10.0 | 43.7 |
| | Qwen-7B | EX | 82.3 | 81.4 | 59.8 | 55.4 | 73.8 |
| | | EM | 36.7 | 34.5 | 17.8 | 3.6 | 27.3 |
| | XiYanSQL-14B | EX | 94.8 | 89.5 | 76.4 | 68.1 | 85.1 |
| | | EM | 91.5 | 82.7 | 63.2 | 54.2 | 77.0 |
| | Llama-8B | EX | 82.3 | 80.9 | 59.2 | 55.4 | 73.5 |
| | | EM | 35.8 | 35.3 | 20.5 | 3.7 | 27.9 |
| ACT-SQL | GPT-4o-mini | EX | 87.9 | 81.4 | 61.5 | 42.8 | 73.4 |
| | | EM | 75.8 | 42.6 | 37.9 | 13.3 | 45.1 |
| | Deepseek-v3 | EX | 91.1 | 83.6 | 56.9 | 50.0 | 75.5 |
| | | EM | 33.5 | 30.3 | 15.5 | 9.0 | 25.2 |
| | Qwen-7B | EX | 89.9 | 86.3 | 56.9 | 51.2 | 76.6 |
| | | EM | 32.3 | 29.1 | 17.8 | 1.2 | 23.5 |
| | XiYanSQL-14B | EX | 90.3 | 84.3 | 58.6 | 49.4 | 75.8 |
| | | EM | 33.1 | 30.0 | 16.7 | 0.0 | 23.7 |
| | Llama-8B | EX | 89.9 | 85.4 | 61.5 | 48.2 | 76.5 |
| | | EM | 33.9 | 28.3 | 13.2 | 1.8 | 22.8 |
| C3-SQL | GPT-4o-mini | EX | 92.7 | 85.0 | 77.6 | 62.0 | 81.9 |
| | | EM | 80.2 | 43.5 | 35.6 | 18.1 | 46.9 |
| | Deepseek-v3 | EX | 89.9 | 85.4 | 61.5 | 48.2 | 76.5 |
| | | EM | 33.5 | 27.8 | 14.9 | 6.0 | 23.5 |
| | Qwen-7B | EX | 81.9 | 76.7 | 70.7 | 47.6 | 72.3 |
| | | EM | 75.0● | 41.7 | 42.0 | 10.2 | 44.7 |
| | XiYanSQL-14B | EX | 91.9 | 81.6 | 76.4 | 51.2 | 78.3 |
| | | EM | 83.9 | 44.4 | 46.0 | 11.4 | 48.8 |
| | Llama-8B | EX | 82.6 | 76.2 | 69.8 | 45.7 | 71.8 |
| | | EM | 74.2 | 41.5 | 41.9 | 10.2 | 44.4 |
| SchemaRAG | GPT-4o-mini | EX | 91.1 | 88.1 | **83.3** | **72.9** | **85.6** |
| | | EM | 86.7 | 65.1 | **64.0** | 44.4 | 66.8 |
| | Deepseek-v3 | EX | 91.2 | **94.3** | **83.7** | **80.6** | **89.6** |
| | | EM | **87.1** | **82.7** | **63.2** | 54.2 | 75.9 |
| | Qwen-7B | EX | **90.7** | 87.4 | **71.8** | **55.4** | **80.4** |
| | | EM | 74.2 | **56.1** | **46.0** | **22.3** | **53.3** |
| | XiYanSQL-14B | EX | **98.0** | **95.7** | **87.4** | **86.1** | **93.3** |
| | | EM | **97.6** | **89.7** | **82.2** | **75.3** | **88.0** |
| | Llama-8B | EX | **88.7** | **87.0** | **71.8** | **62.0** | **80.8** |
| | | EM | **82.3** | **68.8** | **56.3** | **34.9** | **64.5** |

**Table 7: Comparison of EX and VES with evidence on the BIRD dataset**

| Model | LLM | Metric | SIM | MOD | CHALL | TOTAL |
|---|---|---|---|---|---|---|
| TA-SQL | GPT-4o | EX | 60.54 | 40.86 | 38.19 | 52.43 |
| | | VES | 25.28 | 20.33 | 19.13 | 23.21 |
| | Deepseek-v3 | EX | 62.92 | 49.03 | 36.81 | 56.24 |
| | | VES | 27.67 | 25.39 | 13.86 | 25.66 |
| | Qwen-7B | EX | 43.14 | 19.57 | 12.50 | 33.11 |
| | | VES | 16.96 | 7.50 | 4.56 | 12.92 |
| | XiYanSQL-14B | EX | 47.03 | 24.95 | 19.44 | 37.74 |
| | | VES | 20.42 | 13.97 | 9.34 | 17.42 |
| | Llama-8B | EX | 41.62 | 19.14 | 12.50 | 32.06 |
| | | VES | 15.12 | 7.75 | 4.61 | 11.89 |
| DIN-SQL | GPT-4o | EX | 55.78 | 35.27 | 27.08 | 46.86 |
| | | VES | 22.21 | 19.15 | 9.90 | 20.11 |
| | Deepseek-v3 | EX | 54.70 | 38.28 | 27.08 | 47.11 |
| | | VES | 21.15 | 17.05 | 9.60 | 18.81 |
| | Qwen-7B | EX | 36.86 | 14.84 | 8.33 | 27.50 |
| | | VES | 12.63 | 5.83 | 3.11 | 9.67 |
| | XiYanSQL-14B | EX | 52.43 | 34.62 | 22.22 | 44.18 |
| | | VES | 26.60 | 18.90 | 8.86 | 22.58 |
| | Llama-8B | EX | 32.32 | 20.22 | 14.58 | 26.98 |
| | | VES | 12.76 | 8.61 | 4.00 | 10.67 |
| ACT-SQL | GPT-4o | EX | 56.86 | 35.70 | 25.69 | 47.51 |
| | | VES | 25.77 | 17.14 | 15.14 | 22.15 |
| | Deepseek-v3 | EX | 53.68 | 36.72 | 26.57 | 45.98 |
| | | VES | 21.06 | 17.12 | 9.83 | 18.80 |
| | Qwen-7B | EX | 35.42 | 13.68 | 8.21 | 26.27 |
| | | VES | 11.97 | 6.01 | 4.21 | 9.43 |
| | XiYanSQL-14B | EX | 53.51 | 34.84 | 27.08 | 43.16 |
| | | VES | 24.05 | 15.62 | 11.86 | 20.35 |
| | Llama-8B | EX | 32.32 | 20.22 | 14.58 | 26.98 |
| | | VES | 10.65 | 7.63 | 4.22 | 9.12 |
| C3-SQL | GPT-4o | EX | 54.78 | 36.27 | 26.08 | 44.46 |
| | | VES | 20.21 | 19.15 | 9.50 | 18.86 |
| | Deepseek-v3 | EX | 52.76 | 35.28 | 25.08 | 44.85 |
| | | VES | 21.25 | 16.15 | 8.60 | 18.50 |
| | Qwen-7B | EX | 37.97 | 15.95 | 8.43 | 28.51 |
| | | VES | 12.63 | 5.83 | 3.11 | 9.67 |
| | XiYanSQL-14B | EX | 51.63 | 30.25 | 21.77 | 42.34 |
| | | VES | 26.50 | 17.80 | 7.96 | 22.10 |
| | Llama-8B | EX | 32.42 | 20.42 | 14.39 | 27.08 |
| | | VES | 11.37 | 8.15 | 3.85 | 9.66 |
| **SchemaRAG** | GPT-4o | EX | **76.54** | **60.43** | **47.22** | **68.90** |
| | | VES | **34.76** | **29.74** | **21.40** | **31.98** |
| | Deepseek-v3 | EX | **76.11** | **60.00** | **45.83** | **68.38** |
| | | VES | **33.81** | **29.43** | **18.51** | **31.05** |
| | Qwen-7B | EX | **64.86** | **43.01** | **25.00** | **54.50** |
| | | VES | **29.43** | **21.95** | **7.43** | **25.10** |
| | XiYanSQL-14B | EX | **73.30** | **56.13** | **43.06** | **65.25** |
| | | VES | **34.37** | **38.96** | **21.31** | **34.54** |
| | Llama-8B | EX | **71.24** | **47.31** | **32.64** | **60.37** |
| | | VES | **33.06** | **27.67** | **16.99** | **29.92** |

**Table 8: Ablation study of SchemaRAG on the Spider dataset, ● indicates a result that is higher than the corresponding value of the SchemaRAG**

| Model | LLM | Metric | Easy | Medium | Hard | Extra | All |
|---|---|---|---|---|---|---|---|
| **w/o SchemaLinker** | GPT-4o-mini | EX | 91.1 | 81.1 | 83.3● | 72.9 | 85.6 |
| | | EM | 86.7 | 65.1● | 64.0● | 44.4 | 66.9● |
| | Deepseek-v3 | EX | 92.3● | 89.7 | 77.6 | 71.1 | 85.3 |
| | | EM | 87.1 | 84.8● | 66.1● | 59.6● | 78.1● |
| | Qwen-7B | EX | 87.9 | 81.4 | 74.1● | 59.0● | 78.1 |
| | | EM | 72.6 | 52.9 | 51.7● | 25.9● | 53.1 |
| | XiYanSQL-14B | EX | 97.2 | 94.6 | 87.9● | 81.9 | 92.1 |
| | | EM | 97.2 | 92.6● | 85.6● | 74.7 | 89.7● |
| | Llama-8B | EX | 87.1 | 80.7 | 69.0 | 64.5● | 77.7 |
| | | EM | 83.5● | 61.0 | 50.6 | 35.5● | 60.5 |
| **w/o SAR** | GPT-4o-mini | EX | 88.7 | 84.4 | 69.7 | 61.6 | 79.4 |
| | | EM | 32.2 | 33.0 | 22.5 | 4.1 | 26.7 |
| | Deepseek-v3 | EX | 89.6 | 89.7 | 65.2 | 58.9 | 80.8 |
| | | EM | 27.8 | 27.2 | 5.6 | 4.1 | 20.2 |
| | Qwen-7B | EX | 86.3 | 82.5 | 51.7 | 47.0 | 72.5 |
| | | EM | 37.1 | 26.5 | 9.2 | 6.0 | 22.0 |
| | XiYanSQL-14B | EX | 97.6 | 94.8 | 85.6 | 86.1 | 92.6 |
| | | EM | 97.6 | 90.8 | 79.3 | 78.3● | 88.5 |
| | Llama-8B | EX | 75.4 | 61.4 | 47.7 | 39.2 | 58.9 |
| | | EM | 27.0 | 13.7 | 6.3 | 5.4 | 14.3 |
| **w/o POSG** | GPT-4o-mini | EX | 92.1 | 81.1 | 82.6 | 76.6 | 86.5 |
| | | EM | 85.6 | 64.3 | 62.2 | 45.3 | 65.0 |
| | Deepseek-v3 | EX | 91.6 | 90.7 | 78.6 | 72.5 | 86.6 |
| | | EM | 86.9 | 82.2 | 62.6 | 53.3 | 74.2 |
| | Qwen-7B | EX | 87.8 | 82.5 | 70.9 | 54.2 | 79.5 |
| | | EM | 73.2 | 53.1 | 45.2 | 21.2 | 53.2 |
| | XiYanSQL-14B | EX | 97.2 | 94.6 | 87.1 | 82.3 | 92.6 |
| | | EM | 96.4 | 88.5 | 81.7 | 75.2 | 86.5 |
| | Llama-8B | EX | 87.9 | 82.1 | 69.2 | 61.4 | 79.8 |
| | | EM | 81.5 | 64.3 | 54.4 | 32.2 | 62.2 |
| **SchemaRAG** | GPT-4o-mini | EX | **92.2** | **91.1** | 81.1 | **81.7** | **85.6** |
| | | EM | **92.2** | 62.9 | 62.3 | **48.3** | 66.8 |
| | Deepseek-v3 | EX | 91.2 | **94.3** | **83.7** | **80.6** | **89.6** |
| | | EM | **87.1** | 82.7 | 63.2 | 54.2 | 75.9 |
| | Qwen-7B | EX | **90.7** | **87.4** | 71.8 | 55.4 | **80.4** |
| | | EM | **74.2** | **56.1** | 46.0 | 22.3 | **53.3** |
| | XiYanSQL-14B | EX | **98.0** | **95.7** | 87.4 | **86.1** | **93.3** |
| | | EM | **97.6** | 89.7 | 82.2 | 75.3 | 88.0 |
| | Llama-8B | EX | **88.7** | **87.0** | **71.8** | 62.0 | **80.8** |
| | | EM | 82.3 | **68.8** | **56.3** | 34.9 | **64.5** |

Each step uses short bullets or single-sentence items and inline code formatting for table/column names to make intermediate decisions machine-parsable. After `</think>`, the response gives a concise natural-language summary and explicitly marks the final selected field(s) in square brackets (e.g., `[head.age]`), enabling automatic extraction and consistency checking against the ground-truth SQL. During the second stage of multi-task fine-tuning, for the error identification task, the prompt–output pairs were formatted as shown in Figure 4. For the error correction task, the prompt–output pairs were formatted as shown in Figure 5. For the error identification task. The types of errors corrected in this stage are multifaceted and include:

- **Schema Interpretation Errors:** This includes mistakes such as using incorrect or non-existent schema element names, for example writing `department.Creation` instead of the correct `department.creation`.
- **Incomplete Relational Analysis:** The model learns to identify when the analysis overlooks crucial linking tables (e.g., the `management` table) or fails to include all fields necessary for establishing table relationships (e.g., `management.head_ID`).
- **Omission of Filtering Conditions:** Errors where key filtering criteria are missed, such as failing to filter based on the value in `management.temporary_acting`, are also corrected.

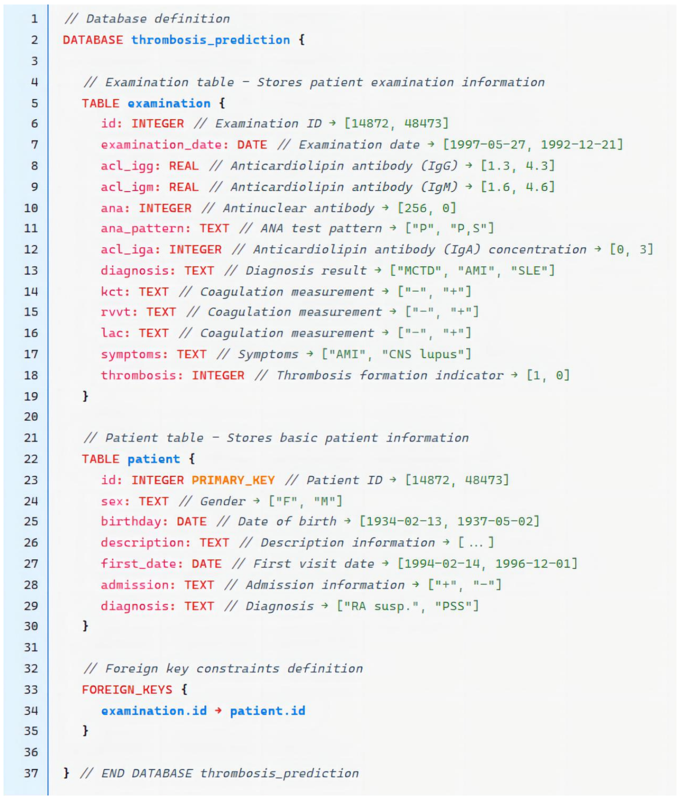
The output format is tailored to the specific task:

- **Error Identification (Figure 4):** The expected output is a step-by-step *Error Analysis* that explicitly details each mistake found in the initial reasoning process.

**Table 9: Ablation study of SchemaRAG on the BIRD dataset, ● indicates a result that is higher than the corresponding value of the SchemaRAG**

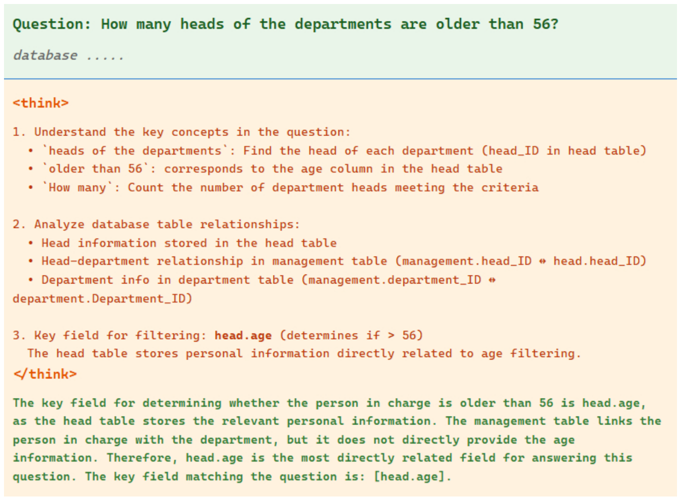
| Model | LLM | Metric | SIM | MOD | CHALL | TOTAL |
|---|---|---|---|---|---|---|
| **w/o SchemaLinker** | GPT-4o | EX | 72.22 | 56.77 | 47.22 | 65.19 |
| | | VES | 32.63 | 29.52 | 23.20● | 30.80 |
| | Deepseek-v3 | EX | 72.11 | 56.13 | 45.14 | 64.73 |
| | | VES | 32.92 | 29.89● | 21.55● | 30.93 |
| | Qwen-7B | EX | 64.54 | 43.66● | 32.64● | 55.22 |
| | | VES | 31.03● | 26.07● | 17.06● | 28.22● |
| | XiYanSQL-14B | EX | 70.27 | 56.13 | 41.67 | 63.30 |
| | | VES | 31.04 | 34.85 | 21.65● | 31.32 |
| | Llama-8B | EX | 65.08 | 43.87 | 29.86 | 54.50 |
| | | VES | 30.79 | 29.59● | 15.06 | 28.95 |
| **w/o SAR** | GPT-4o | EX | 61.41 | 41.29 | 31.94 | 52.54 |
| | | VES | 27.26 | 19.47 | 16.50 | 23.89 |
| | Deepseek-v3 | EX | 60.86 | 40.86 | 31.94 | 52.09 |
| | | VES | 25.00 | 18.09 | 15.35 | 22.00 |
| | Qwen-7B | EX | 55.78 | 35.27 | 25.00 | 46.68 |
| | | VES | 23.07 | 21.22 | 9.01 | 21.19 |
| | XiYanSQL-14B | EX | 69.30 | 53.76 | 41.67 | 61.99 |
| | | VES | 30.47 | 29.25 | 17.45 | 28.88 |
| | Llama-8B | EX | 55.14 | 35.27 | 26.39 | 46.41 |
| | | VES | 21.62 | 20.65 | 8.78 | 20.12 |
| **w/o POSG** | GPT-4o | EX | 73.24 | 58.68 | 47.02 | 66.80 |
| | | VES | 32.52 | 29.38 | 21.23 | 30.88 |
| | Deepseek-v3 | EX | 74.31 | 58.73 | 45.33 | 66.28 |
| | | VES | 31.92 | 29.11 | 17.50 | 31.00 |
| | Qwen-7B | EX | 65.22 | 42.66 | 26.78 | 54.12 |
| | | VES | 29.98 | 19.88 | 9.24 | 26.23 |
| | XiYanSQL-14B | EX | 71.54 | 55.23 | 42.57 | 65.04 |
| | | VES | 32.24 | 35.75 | 20.42 | 33.63 |
| | Llama-8B | EX | 66.24 | 43.87 | 31.86 | 59.26 |
| | | VES | 32.22 | 26.55 | 15.72 | 29.82 |
| **SchemaRAG** | GPT-4o | EX | **76.54** | **60.43** | **47.22** | **68.90** |
| | | VES | **34.76** | **29.74** | 21.40 | **31.98** |
| | Deepseek-v3 | EX | **76.11** | **60.00** | **45.83** | **68.38** |
| | | VES | **33.81** | 29.43 | 18.51 | **31.05** |
| | Qwen-7B | EX | **65.95** | 43.01 | 27.08 | **55.35** |
| | | VES | 30.58 | 20.08 | 9.58 | 25.10 |
| | XiYanSQL-14B | EX | **73.3** | **56.13** | **43.06** | **65.25** |
| | | VES | **34.37** | **38.96** | 21.31 | **34.54** |
| | Llama-8B | EX | **71.24** | **47.31** | **32.64** | **60.37** |
| | | VES | **33.06** | 27.67 | **16.99** | **29.92** |

- **Error Correction (Figure 5):** The output begins with a `<think>` block that outlines the errors. This is followed by a corrected narrative explaining the proper query logic. The output concludes with a final, corrected list of the key fields required to answer the question.



```
// Database definition
DATABASE thrombosis_prediction {

  // Examination table - Stores patient examination information
  TABLE examination {
    id: INTEGER // Examination ID → [14872, 48473]
    examination_date: DATE // Examination date → [1997-05-27, 1992-12-21]
    acl_igg: REAL // Anticardiolipin antibody (IgG) → [1.3, 4.3]
    acl_igm: REAL // Anticardiolipin antibody (IgM) → [1.6, 4.6]
    ana: INTEGER // Antinuclear antibody → [256, 0]
    ana_pattern: TEXT // ANA test pattern → ["P", "P,S"]
    acl_iga: INTEGER // Anticardiolipin antibody (IgA) concentration → [0, 3]
    diagnosis: TEXT // Diagnosis result → ["MCTD", "AMI", "SLE"]
    kct: TEXT // Coagulation measurement → ["-", "+"]
    rvvt: TEXT // Coagulation measurement → ["-", "+"]
    lac: TEXT // Coagulation measurement → ["-", "+"]
    symptoms: TEXT // Symptoms → ["AMI", "CNS lupus"]
    thrombosis: INTEGER // Thrombosis formation indicator → [1, 0]
  }

  // Patient table - Stores basic patient information
  TABLE patient {
    id: INTEGER PRIMARY_KEY // Patient ID → [14872, 48473]
    sex: TEXT // Gender → ["F", "M"]
    birthday: DATE // Date of birth → [1934-02-13, 1937-05-02]
    description: TEXT // Description information → [ ... ]
    first_date: DATE // First visit date → [1994-02-14, 1996-12-01]
    admission: TEXT // Admission information → ["+", "-"]
    diagnosis: TEXT // Diagnosis → ["RA susp.", "PSS"]
  }

  // Foreign key constraints definition
  FOREIGN_KEYS {
    examination.id → patient.id
  }

} // END DATABASE thrombosis_prediction
```

**Figure 2: An example of PromptSchema.**

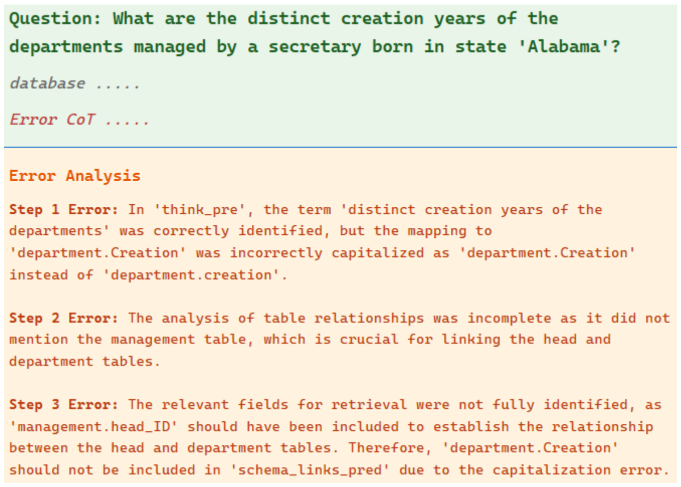


```
Question: How many heads of the departments are older than 56?
database .....

<think>
1. Understand the key concepts in the question:
  • `heads of the departments`: Find the head of each department (head_ID in head table)
  • `older than 56`: corresponds to the age column in the head table
  • `How many`: Count the number of department heads meeting the criteria

2. Analyze database table relationships:
  • Head information stored in the head table
  • Head-department relationship in management table (management.head_ID ↔ head.head_ID)
  • Department info in department table (management.department_ID ↔
department.Department_ID)

3. Key field for filtering: head.age (determines if > 56)
  The head table stores personal information directly related to age filtering.
</think>
The key field for determining whether the person in charge is older than 56 is head.age,
as the head table stores the relevant personal information. The management table links the
person in charge with the department, but it does not directly provide the age
information. Therefore, head.age is the most directly related field for answering this
question. The key field matching the question is: [head.age].
```

**Figure 3: An example of a prompt-CoT-response pair used during the first stage of fine-tuning.**

## I Error Analysis Details

We summarize all error category descriptions in Table 10.



**Question: What are the distinct creation years of the departments managed by a secretary born in state 'Alabama'?**

*database .....*

*Error CoT .....*

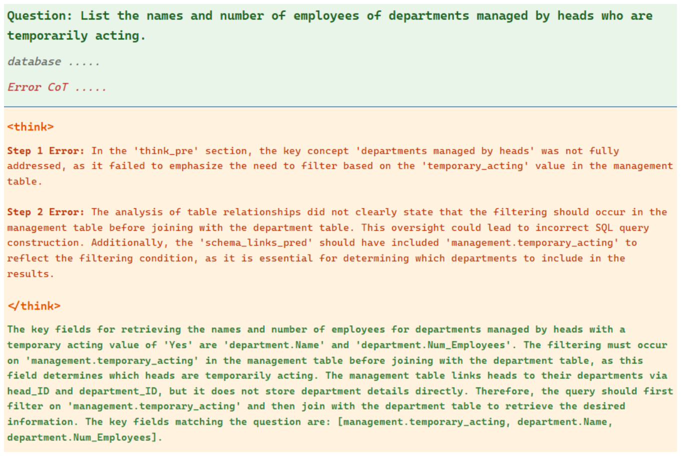
**Error Analysis**

**Step 1 Error:** In 'think_pre', the term 'distinct creation years of the departments' was correctly identified, but the mapping to 'department.Creation' was incorrectly capitalized as 'department.Creation' instead of 'department.creation'.

**Step 2 Error:** The analysis of table relationships was incomplete as it did not mention the management table, which is crucial for linking the head and department tables.

**Step 3 Error:** The relevant fields for retrieval were not fully identified, as 'management.head_ID' should have been included to establish the relationship between the head and department tables. Therefore, 'department.Creation' should not be included in 'schema_links_pred' due to the capitalization error.

**Figure 4: An example of a prompt-response pair used during the second stage of multi-task fine-tuning(Error).**



**Question: List the names and number of employees of departments managed by heads who are temporarily acting.**

*database .....*

*Error CoT .....*

<think>

**Step 1 Error:** In the 'think_pre' section, the key concept 'departments managed by heads' was not fully addressed, as it failed to emphasize the need to filter based on the 'temporary_acting' value in the management table.

**Step 2 Error:** The analysis of table relationships did not clearly state that the filtering should occur in the management table before joining with the department table. This oversight could lead to incorrect SQL query construction. Additionally, the 'schema_links_pred' should have included 'management.temporary_acting' to reflect the filtering condition, as it is essential for determining which departments to include in the results.

</think>

The key fields for retrieving the names and number of employees for departments managed by heads with a temporary acting value of 'Yes' are 'department.Name' and 'department.Num_Employees'. The filtering must occur on 'management.temporary_acting' in the management table before joining with the department table, as this field determines which heads are temporarily acting. The management table links heads to their departments via head_ID and department_ID, but it does not store department details directly. Therefore, the query should first filter on 'management.temporary_acting' and then join with the department table to retrieve the desired information. The key fields matching the question are: [management.temporary_acting, department.Name, department.Num_Employees].

**Figure 5: An example of a prompt-response pair used during the second stage of multi-task fine-tuning(Correct).**

**Table 10: SQL Query Error Types and Examples**

| Error Type | Subcategory | Example of Predicted SQL | Example of Gold SQL |
|---|---|---|---|
| Schema Errors | Wrong Table | `SELECT c.CustomerName,`<br>`  COUNT(*) AS total_orders`<br>`FROM sales_Customers c`<br>`JOIN sales_Orders o`<br>`  ON c.CustomerID = o.CustomerID`<br>`JOIN sales_Invoices i`<br>`  ON c.CustomerID = i.CustomerID`<br>`WHERE o.OrderDate > '2014-01-01'`<br>`GROUP BY c.CustomerName` | `SELECT c.CustomerName,`<br>`  COUNT(*) AS total_orders`<br>`FROM sales_Customers c`<br>`INNER JOIN sales_Orders o`<br>`  ON c.CustomerID = o.CustomerID`<br>`WHERE o.OrderDate > '2014-01-01'`<br>`GROUP BY c.CustomerName` |
| | Wrong Column | `SELECT`<br>`  YEAR(order_purchase_date) AS year,`<br>`  COUNT(*) AS order_count`<br>`FROM orders`<br>`WHERE order_status = 'delivered'`<br>`GROUP BY year` | `SELECT`<br>`  YEAR(order_delivered_date) AS year,`<br>`  COUNT(*) AS order_count`<br>`FROM orders`<br>`WHERE order_status = 'delivered'`<br>`GROUP BY year` |
| | Incorrect Dialect Function | `SELECT location_name,`<br>`  SQRT(POW(lng2-lng1, 2) +`<br>`    POW(lat2-lat1, 2)) AS distance`<br>`FROM locations`<br>`WHERE city = 'New York'` | `SELECT location_name,`<br>`  ST_Distance(point1, point2)`<br>`    AS distance_meters`<br>`FROM locations`<br>`WHERE city = 'New York'` |

| Error Type | Subcategory | Example of Predicted SQL | Example of Gold SQL |
|---|---|---|---|
| Data Analysis Errors | Incorrect Calculation | SELECT department, employee_name,<br>  salary, AVG(salary) OVER ()<br>    AS dept_avg_salary,<br>  RANK() OVER (ORDER BY salary)<br>    AS salary_rank<br>FROM employees | SELECT department, employee_name,<br>  salary,<br>  AVG(salary) OVER<br>    (PARTITION BY department)<br>    AS dept_avg_salary,<br>  RANK() OVER (PARTITION BY<br>    department ORDER BY salary DESC)<br>    AS dept_salary_rank<br>FROM employees |
| | Incorrect Planning | SELECT customer_id,<br>  SUM(order_amount) AS total<br>FROM orders<br>WHERE order_amount > (<br>  SELECT AVG(order_amount)<br>  FROM orders)<br>GROUP BY customer_id<br>ORDER BY total DESC LIMIT 10 | WITH customer_totals AS (<br>  SELECT customer_id,<br>    SUM(order_amount) AS total,<br>    COUNT(*) AS order_count<br>  FROM orders<br>  GROUP BY customer_id),<br>overall_avg AS (<br>  SELECT AVG(order_amount)<br>    AS avg_amount FROM orders)<br>SELECT ct.customer_id, ct.total,<br>  ct.order_count<br>FROM customer_totals ct<br>CROSS JOIN overall_avg oa<br>WHERE ct.total / ct.order_count<br>  > oa.avg_amount<br>ORDER BY ct.total DESC LIMIT 10 |
| Other Errors | Join Error | SELECT inv.InvoiceNumber,<br>  invl.Description, invl.Quantity<br>FROM sales_Invoices inv<br>JOIN sales_InvoiceLines invl<br>  ON inv.OrderID = invl.InvoiceID<br>WHERE inv.InvoiceDate > '2024-01-01' | SELECT inv.InvoiceNumber,<br>  invl.Description, invl.Quantity<br>FROM sales_Invoices inv<br>JOIN sales_InvoiceLines invl<br>  ON inv.InvoiceID = invl.InvoiceID<br>WHERE inv.InvoiceDate > '2024-01-01' |
| | Condition Filter Error | SELECT patient_id, diagnosis<br>FROM medical_records<br>WHERE diagnosis_code IN<br>  ('diabetes', 'hypertension')<br>  AND visit_date > '2024-01-01' | SELECT patient_id, diagnosis<br>FROM medical_records<br>WHERE diagnosis_name IN<br>  ('Type 2 Diabetes',<br>   'Essential Hypertension')<br>  AND visit_date > '2024-01-01' |

## J Automation Scope

### J.1 Workflow Automation and Manual Components

**Table 11: Automation Scope and Manual Involvement in SchemaRAG**

| Component Type | Specific Task / Module and Description |
|---|---|
| **Automated** | **PromptSchema**: Automatically extracts representative data samples using the BM25S algorithm.<br>**SchemaRAG**: The complete end-to-end SQL generation pipeline.<br>**CoT Data Validation**: An automated pipeline to validate the format, consistency, and plausibility of CoT data.<br>**RAG Dataset Validation**: An automated pipeline to verify SQL structural consistency and semantic matching. |
| **Manual** | **Error Dataset Curation**: Manually curating a refined set of error samples for the multi-task learning stage.<br>**Experimental Error Analysis**: Post-run analysis by researchers for the paper (not an operational model component). |

### J.2 Automated CoT Data Validation Pipeline

To ensure the quality and consistency of the CoT data used for model training, we designed and implemented an automated data generation and validation pipeline, is engineered to filter out non-compliant or logically inconsistent reasoning generated by the LLM.

Our automated process begins by invoking an LLM to generate a CoT trace, using the natural language question, database schema, and the ground-truth SQL query as input (`generate_cot` function). Subsequently, each generated CoT output must pass a series of strict validation checks (`validate_cot` function), detailed as follows:

(1) **Format Compliance Check**: We first employ regular expressions to strictly verify that the CoT text adheres to our predefined template (`validate_cot_format` function). This check ensures the output:
- Contains all required analytical steps, such as "1. Understand the key concepts...", "2. Analyze database table relationships...", and "3. Key field for filtering...".
- Includes the final conclusive statement: "The key field matching the question is: [...]".

Any data failing this structural validation is immediately discarded.

(2) **Schema-Label Consistency Check**: This is the core quality check. We define the "Ground-Truth Label" (`ground_truth_label`) as the set of tables and columns extracted from the ground-truth SQL query (using the `sqlparse` library). Concurrently, we parse the "Predicted Label" (`predicted_label`) from the generated CoT text. We enforce a strict match between the `predicted_label` and the `ground_truth_label`, ensuring the CoT's reasoning process strictly corresponds to the entities used in the SQL solution.

(3) **Entity and Key Field Plausibility Check**: We perform further plausibility checks to filter out logically incomplete reasoning. These checks require that:
- At least one "Key Field" (e.g., `head.age`) must be successfully parsed from the CoT text.
- The set of tables mentioned in the CoT (`cot_tables`) must have at least one table in common (a non-empty intersection) with the tables used in the ground-truth SQL (`sql_tables`). This prevents the model from "hallucinating" reasoning that is entirely disconnected from the actual SQL query.

Within the `process_dataset` function, we execute this entire pipeline over our dataset. Only the CoT samples that successfully pass all three validation stages are marked as "validated" and saved to the final training set (`data_cot.json`). This rigorous process guarantees that our fine-tuning dataset possesses high fidelity in both its format and logical accuracy.

### J.3 Automated RAG Dataset Validation Pipeline

Our automated process begins by invoking an LLM (GPT-4o) to generate example "Question–SQL–Schema" triples with structural similarity, using the original question, ground-truth SQL query, and database schema as input (`generate_rag_example` function). Subsequently, each generated RAG example must pass a series of strict validation checks (`validate_rag_example` function), detailed as follows:

(1) **Structural Consistency Check**: We employ a custom script to verify the structural consistency between the new SQL query and its original counterpart (`validate_structural_consistency` function). This check ensures the output:
- Maintains similar SQL structure, including clauses (e.g., SELECT, FROM, WHERE, JOIN), aggregations, and overall query patterns.

Any data failing this structural validation is immediately discarded.

(2) **Semantic Matching Check**: We utilize Claude 4 Sonnet to perform an automated semantic matching check (`validate_semantic_matching` function). This ensures that the generated natural language question accurately corresponds to its SQL query, confirming semantic alignment without hallucinations or deviations.

Within the `process_dataset` function, we execute this entire pipeline over the official development sets of the target datasets (e.g., Spider, BIRD, Spider 2.0, Beaver). Only the examples that successfully pass both validation stages are marked as "validated" and saved to the final extended RAG datasets (e.g., `RAG_Spider.json`, `RAG_BIRD.json`). This rigorous process guarantees that our RAG-augmented datasets possess high fidelity in both their structural integrity and semantic accuracy, enabling robust generalization across diverse SQL query types and database structures.

## K The limition of BM25S

To analyze the limitations of BM25 in handling database values that require deep semantic knowledge, we manually constructed a challenge set of 50 queries from UniSQL. These queries explicitly require reasoning over specific database values to generate the correct SQL.

We first evaluated the overall value coverage of BM25 on this challenge set. As summarized in Table 12, BM25 achieved full coverage for 38 queries (76.0%) but failed to retrieve all necessary values for 12 queries (24.0%). The average coverage across the entire set was 79.3%.

**Table 12: Overall Performance Summary**

| Metric | Value |
|---|---|
| Queries with Full Coverage | 38 (76.0%) |
| Queries with Missing Values | 12 (24.0%) |
| Average Coverage | 79.3% |

Table 13 provides a more detailed distribution of these coverage rates. Notably, among the 24% of queries with imperfect coverage, a significant portion (16.0% of the total set, or 8 queries) had 0% coverage.

Table 13: Distribution of Coverage Rates

| Coverage Range | Number of Queries | Percentage |
|---|---|---|
| 100% (Perfect) | 38 | 76.0% |
| 50% - 99% | 3 | 6.0% |
| 1% - 49% | 1 | 2.0% |
| 0% (Complete Failure) | 8 | 16.0% |
| **Total** | **50** | **100%** |

To understand the root causes of these retrieval failures, we qualitatively analyzed the 12 queries with less than 100% coverage, as detailed in Table 14.

Table 14: Queries with Missing Values (Coverage < 100%)

| Question | Coverage | Missing |
|---|---|---|
| List the long names of buildings constructed before 1950... | 0.0% | 3 values |
| List the name and floor of the building with the largest floor number | 33.3% | 2 values |
| What is the name of the building, total number of subjects... for each physical IAP session location? | 0.0% | 2 values |
| What is total number of subjects, the total fee... for all virtual IAP sessions? | 0.0% | 2 values |
| For each term, list the term code, the term description... | 33.3% | 4 values |
| What is the author, school name, material status... | 50.0% | 1 value |
| List all buildings with their names, building numbers... | 0.0% | 1 value |
| For the 2023 Fall term, what are the unique term descriptions... | 0.0% | 1 value |
| What are the unique titles of subjects offered in the fall term... | 0.0% | 1 value |
| List the building names, names of HR departments... | 0.0% | 1 value |
| Retrieve the titles of subjects offered in the summer term... | 50.0% | 1 value |
| For each distinct mailing list containing more than 1000 people... | 0.0% | 2 values |

This analysis revealed that BM25's failures are not random but fall into four distinct categories:

(1) **Semantic Gap**: BM25's purely lexical matching fails when database values require semantic interpretation (e.g., `%FA` meaning "Fall term").
(2) **Technical Patterns**: Format strings, regex patterns, and special characters are poorly handled by term-based retrieval.
(3) **Numeric Reasoning**: Threshold values and numeric constraints (e.g., "before 1950" or "more than 1000") require domain knowledge beyond lexical matching.
(4) **Domain-Specific Encoding**: Database values often use abbreviated or encoded representations (e.g., date formats or status codes) that differ from natural language queries.

This result indicates that BM25 struggles with technical patterns, semantic encodings, and domain-specific values, leading to a 24% failure rate and only 79.3% average coverage. Its limitations are most evident in handling date formats, pattern strings, and numeric thresholds, indicating significant gaps in bridging the semantic mismatch between natural language queries and database value representations.