JOLT-SQL: Joint Loss Tuning of Text-to-SQL with Confusion-aware Noisy Schema Sampling

Jinwang Song[1], Hongying Zan[1]*, Kunli Zhang[1], Lingling Mu[1], Yingjie Han[1], Haobo Hua[2], Min Peng[3],
[1]Zhengzhou University, [2]Zhengzhou University of Aeronautics, [3]Wuhan University,
jwsong@gs.zzu.edu.cn, iehyzan@zzu.edu.cn

Abstract

Text-to-SQL, which maps natural language to SQL queries, has benefited greatly from recent advances in Large Language Models (LLMs). While LLMs offer various paradigms for this task, including prompting and supervised fine-tuning (SFT), SFT approaches still face challenges such as complex multi-stage pipelines and poor robustness to noisy schema information. To address these limitations, we present JOLT-SQL, a streamlined single-stage SFT framework that jointly optimizes schema linking and SQL generation via a unified loss. JOLT-SQL employs discriminative schema linking, enhanced by local bidirectional attention, alongside a confusion-aware noisy schema sampling strategy with selective attention to improve robustness under noisy schema conditions. Experiments on the Spider and BIRD benchmarks demonstrate that JOLT-SQL achieves state-of-the-art execution accuracy among comparable-size open-source models, while significantly improving both training and inference efficiency. Our code is available at https://github.com/Songjw133/JOLT-SQL.

1 Introduction

Text-to-SQL technology, which transforms natural language questions into executable SQL queries (Deng et al., 2022), aims to break down the technical barriers for users interacting with complex databases, enabling non-professionals to conveniently access desired data. With the widespread adoption of data-driven applications across various domains, the research value and application prospects of Text-to-SQL have become increasingly significant. In recent years, the rise of Large Language Models (LLMs) has injected new vitality into this field, and LLM-based Text-to-SQL methods have made remarkable progress (Hong et al., 2024; Shi et al., 2025).

Currently, LLM-based Text-to-SQL methods can be broadly categorized into two main paradigms: Prompting approaches, including methods like In-Context Learning (ICL) and Chain-of-Thought (CoT) (Wei et al., 2022; Dong et al., 2024); and Supervised Fine-Tuning (SFT). Prompting methods have garnered considerable attention due to their impressive performance in zero-shot or few-shot scenarios. However, they often rely on powerful, proprietary commercial models, and their effectiveness is typically difficult to replicate with smaller-parameter open-source models.

In contrast, SFT offers a more controllable path to enhance open-source models. Task-specific SFT significantly improves SQL generation capabilities and provides practical advantages such as lower deployment costs, offline operation, and support for local data processing.

Schema linking is a crucial component in mainstream Text-to-SQL methods (Yang et al., 2024c). Its purpose is to select the necessary database tables/columns and filter out database schema elements irrelevant to the user's question, thereby reducing interference and improving SQL generation performance. CHESS (Talaei et al., 2024) employs a dedicated agent to prune irrelevant schema elements. E-SQL (Caferoğlu and Ulusoy, 2024) optimizes the schema linking phase through question enrichment and candidate predicate expansion. DIN-SQL (Pourreza and Rafiei, 2023) enhances the model's schema linking capabilities using decomposed ICL.

In existing approaches, schema linking is often treated as a separate step from SQL generation. In SFT-based methods, schema linking and SQL generation tasks are typically trained separately (Pourreza and Rafiei, 2024), leading to increased task costs. In contrast, ROUTE (Qin et al., 2024) utilizes multi-task learning to incorporate schema linking, SQL generation, and other sub-tasks into a single SFT process. However, this approach also

*Corresponding author.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 6040–6053
November 4-9, 2025

means that the total volume of training data increases linearly with the number of tasks, introducing additional training overhead.

In this work, we propose a novel method: **JO**int **L**oss **T**uning of Text-to-SQL with Confusion-aware Noisy Schema Sampling (JOLT-SQL). Our objective is to address two primary issues: (1) Conventional pipeline fine-tuning strategies or multi-task learning approaches often require fine-tuning multiple distinct models or incur increased training time due to the expanded multi-task dataset, leading to higher temporal and hardware costs. (2) During actual inference, the schema information provided by the schema linking module is often imperfect, potentially containing irrelevant items or omitting necessary ones. Existing SQL generation models, when fine-tuned solely on ground truth schema or with random sampled noisy schema, struggle to adapt to such imperfect inputs during inference, thereby affecting SQL generation accuracy.

JOLT-SQL addresses these challenges through an innovative joint loss tuning strategy. Specifically, by adjusting attention masks, JOLT-SQL stands as the first LLM SFT method to fuse the optimization objectives of schema linking and SQL generation tasks within a single backward pass. Concurrently, the method dynamically samples noisy schema based on probability during training, enhancing its ability to handle imperfect schema linking results during inference.

Our contributions are as follows:

• We propose a discriminative schema linking fine-tuning method that incorporates local bidirectional attention. Experiments demonstrate that this method achieves state-of-the-art performance in schema linking on the Spider and BIRD datasets, along with faster inference speeds.

• Unlike typical pipeline fine-tuning strategies, we train schema linking and SQL generation using joint loss, thereby avoiding the additional overhead associated with multi-stage fine-tuning.

• We further introduce Confusion-aware Noisy Schema Sampling, enhancing model robustness against redundant schema linking results. This technique strategically guides the SQL generation task's attention by incorporating noisy schema items selected based on the model's points of predictive confusion. Experimental results show that JOLT-SQL, using the Qwen2.5-Coder-14B model, achieves 88.4%/88.9% execution accuracy on the Spider Dev/Test set and 64.9% on the BIRD Dev set.

2 Related Work

Large Language Models (LLMs) have shown significant promise in Text-to-SQL, with research largely following two paths: Prompting and Supervised Fine-Tuning (SFT).

2.1 Prompting for Text-to-SQL

These methods guide LLMs with specific instructions. DIN-SQL (Pourreza and Rafiei, 2023) used decomposed in-context learning and a multi-stage pipeline. DAIL-SQL (Gao et al., 2024a) focused on example selection for complex scenarios. Other works refined prompting and schema understanding: C3 (Dong et al., 2023) improved schema representation and used a multi-module pipeline, while MAC-SQL (Wang et al., 2025) employed multi-agent collaboration. To handle increased complexity, CHESS (Talaei et al., 2024) utilized agent-based schema pruning and multi-turn dialogue; E-SQL (Caferoğlu and Ulusoy, 2024) optimized schema linking via question enrichment and candidate predicate expansion; and RSL-SQL (Cao et al., 2024) applied bi-directional pruning methods to improve schema linking recall. The effectiveness of these approaches remains highly dependent on prompt design and the LLM's inherent understanding capabilities.

2.2 Supervised Fine-tuning based Text-to-SQL

SFT aims to more deeply integrate task-specific knowledge into open-source LLMs. While early methods involved direct language model fine-tuning, later strategies became more specialized. DTS-SQL and DB-Explore (Pourreza and Rafiei, 2024; Ma et al., 2025) treated schema linking as a separate fine-tuning stage, whereas ROUTE (Qin et al., 2024) used multi-task learning for schema linking and SQL generation among other sub-objectives. ExSL (Glass et al., 2025) proposed an efficient extractive schema linking method for decoder-only LLMs. Other approaches include SENSE (Yang et al., 2024b), which introduced reinforcement learning, and SQL-PaLM (Sun et al., 2023) and CodeS (Li et al., 2024), which enhanced models via secondary pre-training on large SQL corpora. BASE-SQL (Sheng et al., 2025) also adopted a multi-stage fine-tuning framework. These SFT methods have proven to be effective pathways for improving Text-to-SQL performance on open-source models.

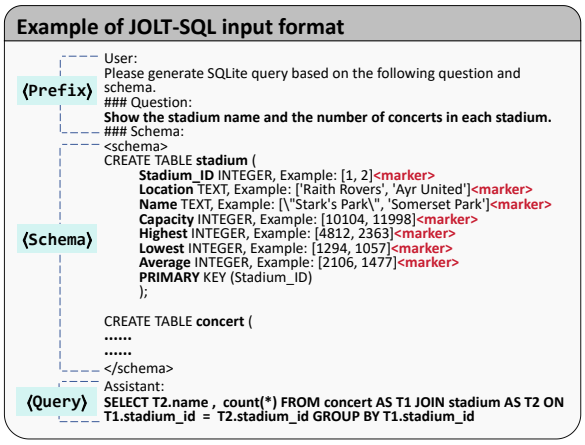


Figure 1: An input example of JOLT-SQL.

3 Approach

In this section, we detail our proposed JOLT-SQL method. We begin by describing its unique schema formatting, followed by an exploration of the joint loss design for schema linking and SQL generation tasks.

3.1 Schema Representation Standardization

We first standardize the schema representation to ensure a uniform input format for subsequent processing stages.

In JOLT-SQL, we divide the input sequence during the training phase into three distinct parts based on content: ⟨Prefix⟩, ⟨Schema⟩, and ⟨Query⟩, as illustrated in Figure 1. Specifically:

• ⟨Prefix⟩: Contains essential task instructions and the user's question.

• ⟨Schema⟩: The schema section, describing the complete table structure and column definitions.

• ⟨Query⟩: The ground truth SQL query statement part of each input sequence, used for supervised training.

We employ a Data Definition Language (DDL)-style representation for the schema (Gao et al., 2024b). For each table in the database, we specify the name and data type of every column, along with its primary and foreign key information. Additionally, for each column, we select at most two value examples from the database to include in its definition.

After each column definition within ⟨Schema⟩, we insert a special marker token, denoted as ⟨marker⟩. We simply select the model's padding token (e.g., "<|endoftext|>" from the Qwen2 tokenizer) as our marker token. As a special token, it acts as an isolated token that avoids combining with other characters into subwords, thereby simplifying subsequent processing, and is also semantically neutral.

3.2 Discriminative Schema Linking with Local Bidirectional Attention

Unlike typical schema linking methods based on generative LLMs, and inspired by LS-LLaMA (Li et al., 2023b), our approach relies on directly discriminating the hidden states of marker tokens.

Specifically, given a complete input sequence $X = \langle \texttt{Prefix}, \texttt{Schema}, \texttt{Query} \rangle = [x_1, x_2, \ldots, x_n]$, after it passes through the LLM decoder layers, we obtain its hidden states $H \in \mathbb{R}^{n \times d}$. For the hidden state h_i corresponding to the i-th token, we pass it through a linear layer $W \in \mathbb{R}^{1 \times d}$ and a sigmoid function to convert it into a probability:

$$\hat{y}_i = \sigma(W h_i) \tag{1}$$

Concurrently, for each input sequence X, we create a binary mask $M = [m_1, m_2, \ldots, m_n]$. An element m_i is set to 1 if the i-th token is a marker token located within the ⟨Schema⟩, and 0 otherwise. The loss is defined as:

$$\mathcal{L}_{\text{SL}} = -\frac{1}{\sum_{i=1}^{n} m_i} \sum_{i=1}^{n} m_i \cdot \text{BCE}(\hat{y}_i, y_i) \tag{2}$$

Here, BCE means binary cross-entropy loss, $y_i \in \{0, 1\}$ is the ground truth label for the schema linking task.

Furthermore, decoder-only LLM architectures employ a causal attention mask, which prevents tokens within ⟨Schema⟩ from accessing global schema information. As an improvement, we enable tokens within the ⟨Schema⟩ part to additionally have local bidirectional attention, on top of the default causal attention. Let $A(x_i)$ denote the set of indices of other tokens that x_i can attend to, and I denote the set of indices for token positions in the sequence, $A(x_i)$ is defined as:

$$\forall i \in I_{\langle \texttt{Schema} \rangle},\ A(x_i) = \begin{cases} (I_{\langle \texttt{Prefix} \rangle} \cup I_{\langle \texttt{Schema} \rangle}) \setminus I_{\langle \texttt{marker} \rangle}, & i \notin I_{\langle \texttt{marker} \rangle} \\ I_{\langle \texttt{Prefix} \rangle} \cup I_{\langle \texttt{Schema} \rangle}, & i \in I_{\langle \texttt{marker} \rangle} \end{cases} \tag{3}$$

It should be noted that we introduce a further adjustment rule for marker tokens: marker tokens are invisible to all non-marker tokens. This modification allows us to remove marker tokens during the SQL generation stage.

3.3 SQL Supervised Tuning with Schema Selective Attention

For the SQL query generation task, we enhance the model performance by applying supervised tuning to the ⟨Query⟩ part. A challenge here is that during training, the input sequence contains the full ⟨Schema⟩. For pipeline SFT methods, the SQL generation model is often fine-tuned on the ground truth schema subset, which helps to maintain consistency with the input conditions expected during inference (Pourreza and Rafiei, 2024).

We address this issue by similarly adjusting the attention mask. Specifically, tokens in the ⟨Query⟩ part selectively attend only to tokens within the ground truth schema subset (referred to as ⟨GT_Schema⟩), rather than the full ⟨Schema⟩. In practice, ⟨GT_Schema⟩ includes not only the relevant column definitions but also the table structure to which each column definition belongs (such as table names, primary key, and foreign key definitions).

In addition to ⟨GT_Schema⟩, to make the model more robust to imperfect schema linking results, we select some noisy schema items (referred to as ⟨Noisy_Schema⟩) for the ⟨Query⟩ tokens to attend to. This attention mechanism can be represented as:

$$\forall i \in I_{\langle \texttt{Query} \rangle}, A(x_i) = \big(I_{\langle \texttt{Prefix} \rangle} \cup I_{\langle \texttt{GT\_Schema} \rangle} \cup I_{\langle \texttt{Noisy\_Schema} \rangle} \cup \{j | j \in I_{\langle \texttt{Query} \rangle}, j \leq i\}\big) \setminus I_{\langle \texttt{marker} \rangle} \quad (4)$$

Combined with the schema Local Bidirectional Attention described in Section 3.2, the final attention mask form is shown in Figure 2.

Consistent with typical LLM SFT, we then calculate the Next Token Prediction (NTP) loss for the SQL query. For a $\langle \texttt{Query} \rangle = [x_{n-m+1}, \ldots, x_n]$, the loss is:

$$\mathcal{L}_{\text{NTP}} = -\frac{1}{m} \sum_{i=n-m+1}^{n} \log P_{A(x_i)}\left(x_i | \langle \texttt{Prefix}, \texttt{Schema}, x_{n-m+1:i-1} \rangle\right) \quad (5)$$

where $P_{A(\cdot)}(\cdot|\cdot)$ denotes the conditional probability given the application of the attention $A(\cdot)$.

3.4 Joint Loss Tuning with Confusion-aware Noisy Schema Sampling

In the preceding sections, we have introduced the respective losses for the schema linking task $\mathcal{L}_{\text{SL}}$ and the SQL generation task $\mathcal{L}_{\text{NTP}}$. In JOLT-SQL, we train the model using a joint loss:

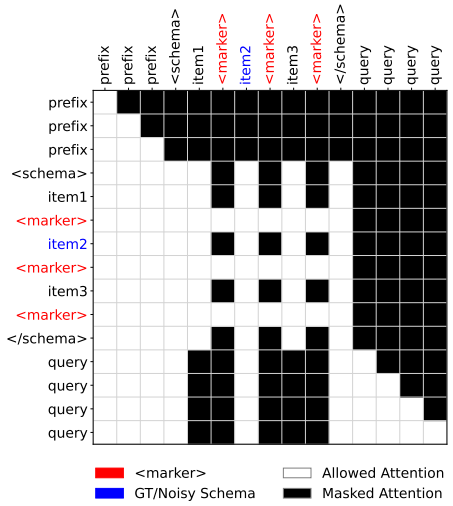


Figure 2: Visualize the attention mask of JOLT-SQL using a sample text. For clarity, the text has been simplified and does not reflect the actual tokenization.

Pseudo-code

```
def train_step(model, input_ids, attn_mask, gt_schema_idx):
#args:
##model: LLM for training.
##input_ids: Token ids of input sequence.
##attn_mask: Vanilla causal attention mask.
##gt_schema_idx: Ground truth schema idx from input_ids.
    with torch.inference_mode():
        attn_mask = apply_schema_bidirectional_attn(input_ids, attn_mask)
        outputs = model.forward(input_ids, attn_mask) #extra forward pass
        sampling_weights = torch.sigmoid(outputs.logits_sl)
        noisy_schema_idx = noisy_schema_sampling(input_ids, sampling_weights)
        selected_schema_idx = set(noisy_schema_idx) | set(gt_schema_idx)

    attn_mask = apply_schema_selective_attn(selected_schema_idx, attn_mask)
    outputs = model.forward(input_ids, attn_mask)
    loss_sl = compute_sl_loss(outputs)
    loss_ntp = compute_ntp_loss(outputs)
    loss = loss_sl + loss_ntp
    loss.backward()
```

Figure 3: Pseudo-code for the JOLT-SQL training process. In the actual experiments, the `extra forward pass` is performed only during the first training epoch, and the `sampling_weights` are cached.

$$\mathcal{L} = \mathcal{L}_{\text{SL}} + \mathcal{L}_{\text{NTP}} \quad (6)$$

As described in Section 3.3, we introduce attention to ⟨Noisy_Schema⟩. Instead of simple random sampling, we employ a probability-weighted sampling to select these noisy schema items. The motivation here is that selecting noisy schema items that the model tends to misidentify with high confidence can help the model better adapt to schema linking results that may contain numerous False Positives during inference.

We define a func $Sample(Sets, Weights, Count)$, which performs sampling without replacement of $Count$ items from the $Sets$ based on their $Weights$. Let $\mathcal{S}$ denote the set of schema items, then:

$$\mathcal{S}_{\langle \texttt{Noisy\_Schema} \rangle} = Sample(\mathcal{S}_{\langle \texttt{Schema} \rangle} \setminus \mathcal{S}_{\langle \texttt{GT\_Schema} \rangle},\ \hat{y},\ \lfloor k \rfloor)$$
$$k \sim U\big(0,\ \lfloor \beta \cdot |\mathcal{S}_{\langle \texttt{Schema} \rangle}| \rfloor\big) \tag{7}$$

Here, β is a float in the range (0,1) that controls the upper bound for the number of samples k. In our experiments, β is set to 0.2 for the Spider dataset and 0.1 for the BIRD dataset.

Thanks to our joint loss tuning strategy, probability-weighted sampling can be efficiently integrated into the training process. During training, this probability-weighted sampling is guided by the model's own propensity for confusion: its predicted probabilities $\hat{y}$ (Equation 1) for marker tokens are dynamically used as the sampling weights. These weights are obtained through an extra forward pass that does not require gradient computation. The pseudo-code for the training process is shown in Figure 3.

Notably, these sampling weights are calculated and cached during the first training epoch and reused in subsequent epochs. The rationale for this strategy is our aim to capture the model's probability distribution when encountering "new" data, which more closely reflects its behavior in actual inference scenarios. In the first epoch, all training data is unseen by the model. In subsequent epochs, as the model progressively fits the training data, its prediction confidence on this seen data increases rapidly, causing the predicted probabilities to become extreme (approaching 0 or 1). This deviates from the model's behavior on unseen data (dev/test sets).

Furthermore, this caching strategy significantly improves training efficiency. The additional time cost incurred by the extra forward pass is minimal. For a more detailed comparison of time efficiency, please refer to Appendix A.1.

3.5 Inference

During schema linking inference, we prioritize recall by setting the decision threshold for $\hat{y}$ to 0.05. This emphasis on high recall is crucial, as missing necessary schema items is more detrimental to subsequent SQL generation than including some redundant ones, which models can often partially handle.

Following schema linking, we prune all marker tokens and irrelevant column definitions identified by the linking results. We then generate the SQL query using a vanilla causal attention mask. This ensures compatibility with frameworks such as HuggingFace Transformers' `model.generate()`, which do not support custom attention masks in autoregressive decoding.

4 Experiments

4.1 Setup

Datasets To evaluate the JOLT-SQL method, we conducted experiments on two widely used datasets: Spider (Yu et al., 2018) and BIRD (Li et al., 2023a). **Spider** comprises 200 databases with multiple tables, and its Training/Dev/Test sets contain 7,000/1,034/2,147 question-SQL pairs, respectively. **BIRD** contains 12,751 unique question-SQL pairs, 95 large databases with a total size of 33.4 GB, and its Training/Dev/Test sets include 9,428/1,534/1,789 pairs, respectively. For these two benchmarks, our models are fine-tuned exclusively on their respective Training sets and evaluated on the corresponding Dev and Test sets.

Evaluation Metrics The quality of SQL generation is evaluated by Execution Accuracy (EX), which measures whether the execution results of the generated SQL on the database exactly match the ground truth answers. This directly reflects the model's practical ability to generate correct SQL.

For schema linking, we use Recall and Precision for evaluation. To more comprehensively evaluate the model's overall discriminative ability across different decision thresholds, we also introduce PR-AUC (Area Under the Precision-Recall Curve) and ROC-AUC (Area Under the Receiver Operating Characteristic Curve).

Model We selected the popular open-source Qwen2.5-Coder (Yang et al., 2024a; Hui et al., 2024) series for our experiments, including Qwen2.5-Coder-7B-Instruct and Qwen2.5-Coder-14B-Instruct.

Implementation Details We employ LoRA (Hu et al., 2022) to reduce VRAM requirements. More details can be found in the Appendix B.2.

Baselines In our experiments, we compare recent LLM-based Text-to-SQL baselines in two categories. The Prompting methods include DIN-SQL (Pourreza and Rafiei, 2023), DAIL-SQL (Gao et al., 2024a), CHESS (Talaei et al., 2024), E-SQL (Caferoğlu and Ulusoy, 2024), MCS-SQL (Lee et al., 2025), RSL-SQL (Cao et al., 2024) and MAC-SQL

Methods	Type	Spider Dev EX	Spider Test EX	BIRD Dev EX
DIN-SQL + GPT-4(Pourreza and Rafiei, 2023)	Prompting	82.8	85.3	50.7
DAIL-SQL + GPT-4(Gao et al., 2024a)	Prompting	83.5	86.6	54.8
CHESS + Gemini1.5-Pro(Talaei et al., 2024)	Prompting	-	87.2	68.3
E-SQL + GPT-4o-mini(Caferoğlu and Ulusoy, 2024)	Prompting	-	74.8	61.6
MAC-SQL + GPT-4(Wang et al., 2025)	Prompting	86.8	82.8	59.4
RSL-SQL + GPT-4o(Cao et al., 2024)	Prompting	-	87.9	67.2
MCS-SQL + GPT-4(Lee et al., 2025)	Prompting	89.5	89.6	63.4
CodeS-7B(Li et al., 2024)	SFT	85.4	83.5	57.2
CodeS-15B(Li et al., 2024)	SFT	84.9	85.0	58.5
SENSE-7B(Yang et al., 2024b)	SFT	83.2	83.5	51.8
SENSE-13B(Yang et al., 2024b)	SFT	84.1	86.6	55.5
DTS-SQL + Deepseek-7B(Pourreza and Rafiei, 2024)	SFT	85.5	84.4	55.8
ExSL + Deepseek-Coder-6.7B(Glass et al., 2025)	SFT	82.4	83.0	63.2
ROUTE + Qwen2.5-7B(Qin et al., 2024)	SFT	83.6	83.7	55.9
ROUTE + Qwen2.5-14B(Qin et al., 2024)	SFT	87.3	87.1	60.9
DB-Explore + Qwen2.5-Coder-7B(Ma et al., 2025)	SFT	84.0	-	52.1
BASE-SQL + Qwen2.5-Coder-14B(Sheng et al., 2025)	SFT	86.8	87.9	63.8
JOLT-SQL(**Ours**) + Qwen2.5-Coder-7B	SFT	87.0	86.8	60.4
JOLT-SQL(**Ours**) + Qwen2.5-Coder-14B	SFT	**88.4**	**88.9**	**64.9**

Table 1: Main experimental results. The best results among SFT-based methods are shown in **bold**, and the second-best are underlined.

Methods	Spider Dev				Spider Test				BIRD Dev			
	P	R	ROC	PR	P	R	ROC	PR	P	R	ROC	PR
ExSL + Deepseek-Coder-6.7B(Glass et al., 2025)	-	-	99.76	98.44	-	-	99.70	98.40	-	-	99.38	93.67
JOLT-SQL(**Ours**) + Qwen2.5-Coder-7B	88.09	98.12	99.86	98.70	90.28	**98.65**	99.87	99.10	**76.14**	94.83	99.46	94.31
JOLT-SQL(**Ours**) + Qwen2.5-Coder-14B	**91.07**	**99.03**	**99.91**	**99.29**	**93.74**	98.58	**99.89**	**99.26**	75.12	**96.68**	**99.54**	**95.42**

Table 2: Schema linking results. **P** and **R** represent column Precision and column Recall, respectively, at a decision threshold of 0.05. **ROC** stands for ROC AUC, and **PR** stands for PR AUC. The best results are shown in **bold**.

(Wang et al., 2025). The SFT-based methods comprise CodeS (Li et al., 2024), SENSE (Yang et al., 2024b), DTS-SQL (Pourreza and Rafiei, 2024), ExSL (Glass et al., 2025), ROUTE (Qin et al., 2024), DB-Explore (Ma et al., 2025), and BASE-SQL (Sheng et al., 2025). These baselines cover a range of prompting and fine-tuning strategies for Text-to-SQL.

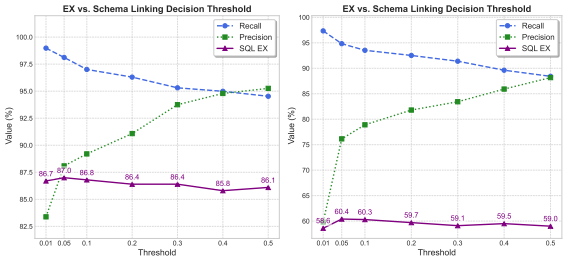
4.2 Main Results

Table 1 presents our main experimental results and a comparison with baselines. JOLT-SQL demonstrates excellent performance on both the Spider and BIRD benchmarks. Specifically, JOLT-SQL using the Qwen2.5-Coder-14B model achieves execution accuracies of 88.4%, 88.9%, and 64.9% on the Spider Dev set, Test set, and BIRD Dev Set, respectively. This performance is state-of-the-art among open-source models of comparable size based on SFT. Notably, JOLT-SQL with Qwen2.5-Coder-7B even surpasses larger models like CodeS-15B and SENSE-13B in results.

Unlike the two-stage fine-tuning strategy of DTS-SQL or the complex multi-stage pipeline of BASE-SQL, JOLT-SQL achieves superior performance and significantly simplifies the training process through its novel single-stage joint loss tuning. Meanwhile, compared to the multi-task learning approach adopted by ROUTE, JOLT-SQL achieves better results while keeping the total amount of training data unchanged. Furthermore, even when compared to prompting methods that often rely on powerful closed-source models like GPT-4 (Achiam et al., 2023; Hurst et al., 2024) or Gemini1.5-Pro (Team et al., 2024), JOLT-SQL demonstrates strong competitiveness, with its performance matching or exceeding some of these top approaches.

4.3 Schema Linking Results

Table 2 displays the schema linking task metrics of our JOLT-SQL method on the Spider Dev/Test



(a) Spider Dev (b) BIRD Dev

Figure 4: Comparison of the model's final EX under different schema linking decision thresholds.

and BIRD Dev sets. We compare it with ExSL (Glass et al., 2025) as a baseline. ExSL employs an extractive method to fine-tune decoder-only LLMs and relies on repeating column definitions to alleviate the issue of incomplete schema information caused by causal masks. In contrast, JOLT-SQL, by introducing a local bidirectional attention mechanism, avoids the need to redundantly duplicate column definitions, which would otherwise increase sequence length. The results show that when using the Qwen2.5-Coder-14B model, JOLT-SQL achieves ROC and PR scores as high as 99.91% and 99.29%, respectively, on the Spider Dev set, and also attains SOTA performance of 99.54% ROC and 95.42% PR on the BIRD Dev set.

We further investigated the impact of the schema linking threshold on the specific performance metrics of JOLT-SQL (based on the Qwen2.5-Coder-7B). In Figure 4, we present the changes in column recall, column precision, and the final SQL EX as the decision threshold decreases from 0.5 to 0.01. On the Spider Dev set, it can be observed that as the decision threshold starts to decrease from 0.5, the EX shows an overall upward trend, reaching its peak EX at a threshold of 0.05. Opting for an even lower threshold of 0.01 results in only a slight improvement in recall, accompanied by a significant sacrifice in precision. This leads to the introduction of excessive False Positive schema items during inference, causing the EX to decline.

BIRD exhibits a similar overall trend with respect to different thresholds, reaching its peak EX around 0.05. Notably, compared to the Spider dataset, an increase in column recall on the BIRD dataset is typically accompanied by a more rapid decline in precision. This might reflect the model's higher uncertainty when dealing with a more complex dataset like BIRD.

4.4 Ablation Study

In the ablation study, we analyzed the impact of three core components on overall performance: schema Local Bidirectional Attention **LBA**, Confusion-aware Noisy Schema Sampling (**Confusion-aware NSS**) and schema Selective Attention **SA**. In the context of these ablations, SA defaults to the ⟨Query⟩ attending only to the ⟨GT_Schema⟩ part of the overall schema information (thus excluding ⟨Noisy_Schema⟩, which would be introduced by the NSS), with NSS treated as an additional component.

The results of the ablation study, detailed in Table 3, confirm that all three core components contribute to JOLT-SQL's performance. LBA was vital for capturing complex schema relationships, and its absence markedly impacted results. NSS proved key for improving robustness against noisy schema during inference by promoting adaptation in training, while SA helped align training with inference conditions by focusing ⟨Query⟩ attention on relevant schema subsets instead of the full schema. The individual or combined removal of these mechanisms consistently reduced performance. Ultimately, disabling all three components substantially degraded overall performance.

The results of the ablation study demonstrate that within our joint loss tuning strategy, the clever integration of these attention adjustment and noise injection mechanisms plays an indispensable role in achieving superior model performance.

Method	Spider Dev EX	BIRD Dev EX
JOLT-SQL + Qwen2.5-Coder-7B	**87.0**	**60.4**
w/o LBA	84.8	58.3
w/o Confusion-aware NSS	86.1	58.6
w/o SA & Confusion-aware NSS	85.9	59.2
w/o LBA & SA & Confusion-aware NSS	84.5	57.7

Table 3: Ablation study results.

4.5 Further Analysis

Impact of Schema LBA The ablation study has already shown that removing LBA leads to a significant decline in model performance. Table 4 further reveals its critical role in schema linking: the absence of LBA causes a deterioration in both recall and precision, posing a dual challenge for SQL generation due to increased noise and omission of relevant columns.

For instance, on the Spider Dev/BIRD Dev sets, after removing LBA, even with a low decision

Methods	Spider Dev				BIRD Dev			
	P	R	ROC	PR	P	R	ROC	PR
JOLT-SQL + Qwen2.5-Coder-7B	**88.09**	**98.12**	**99.86**	**98.70**	**76.14**	**94.83**	**99.46**	**94.31**
w/o LBA	79.83	95.04	99.34	95.41	66.01	92.73	98.86	88.02

Table 4: Ablation results for schema LBA.

No.	Training with(⟨Query⟩ attends to)	Inference with	Spider Dev EX	BIRD Dev EX
#1	Full ⟨Schema⟩	Full ⟨Schema⟩	82.8	52.9
#2	Full ⟨Schema⟩	⟨GT_Schema⟩	89.6	68.5
#3	⟨GT_Schema⟩	⟨GT_Schema⟩	**90.5**	**72.2**
#4	Full ⟨Schema⟩	$\hat{y} > 0.05$ Schema	85.9	59.2
#5	⟨GT_Schema⟩	$\hat{y} > 0.05$ Schema	86.1	58.6
#6	⟨GT_Schema⟩ & Random NSS	$\hat{y} > 0.05$ Schema	86.4	59.6
#7	⟨GT_Schema⟩ & Confusion-aware NSS	$\hat{y} > 0.05$ Schema	87.0	60.4

Table 5: SQL generation performance under different training attention strategies and inference-stage schema input combinations (based on Qwen2.5-Coder-7B, with LBA enabled). $\hat{y} > 0.05$ Schema refers to the results predicted by schema linking. **#1**, **#2**, and **#3** are used for baseline or ideal condition comparisons, with **#3** considered as a performance upper bound. **#4** to **#7** focus on the model's performance under realistic predicted schema conditions, where **#7** represents the complete strategy adopted by JOLT-SQL.

threshold of 0.05, recall dropped to 95.04% and 92.73%, respectively, while precision significantly decreased to 79.83% and 66.01%. The PR also correspondingly decreased from 98.70%/94.31% to 95.41%/88.02%, confirming the weakened ability of the model to identify relevant columns. This is primarily because, under the constraints of the standard causal mask, most marker tokens can only perceive local preceding information and cannot make judgments by synthesizing complete schema information, thus highlighting the necessity of LBA.

Impact of Schema SA The experimental results in Table 5 reveal the role of Schema SA during the training phase. Comparing **#3** (using ⟨GT_Schema⟩ for both training and inference) with **#1** (using Full ⟨Schema⟩ for both), the former, serving as an ideal performance upper bound, significantly outperforms the latter, indicating the necessity of schema linking. **#2** (training with Full ⟨Schema⟩ attention, inference with ⟨GT_Schema⟩ input) shows significant improvement over **#1**, but still falls short of the ideal upper bound of **#3**. This highlights the importance of consistency in input distribution between the training and inference phases. Schema SA achieves this input alignment between training and inference phases by cleverly defining the attention scope.

Impact of NSS In more realistic inference scenarios, the model must effectively handle potentially imperfect schema subsets obtained from the schema linking phase. Experiments **#4** to **#7** in Table 5 explore this aspect. The EX of **#4** (which corresponds to the "*w/o* SA & Confusion-aware NSS" setting in Table 3) can be considered a baseline performance. Subsequently, the EX of **#5** (also the "*w/o* Confusion-aware NSS" setting in Table 3) slightly increased to 86.1% on Spider but dropped to 58.6% on BIRD. This result indicates that although focusing the model on the ideal ⟨GT_Schema⟩ via SA during training is beneficial, a lack of adaptive training for noise still leaves the model insufficiently robust when facing imperfectly predicted schema.

To enhance the model's ability to handle such noise, we introduced the NSS strategy. Experiment **#6** (training with ⟨GT_Schema⟩ + random NSS), compared to **#5**, introduced randomly selected noisy schema items during training, boosting EX on Spider and BIRD to 86.4% and 59.6%, respectively. This preliminarily demonstrates the importance of exposing the model to and training it to handle noise.

Building on this, **#7**, which is the strategy adopted by JOLT-SQL, achieved the best results. **#7** follows the same training procedure as **#6**—both attend to ⟨GT_Schema⟩ while introducing noisy items—but the key difference lies in the selection method for these noisy items: **#6** uses simple random sampling, whereas the confusion-aware approach in **#7** is reflected in its weighted sampling based on the model's predicted probabilities in the training phase. This sampling mechanism tends to prioritize items that the model erroneously assigns a high relevance probability to (potential False Positives). Considering that we intentionally use a low decision threshold during inference to boost recall (which introduces more False Positives), this sampling strategy, focused on learning to handle "confusing" items, is particularly well-suited. By specifically training on schema items prone to model misjudgment, JOLT-SQL can more effectively learn to discern and ignore the most disruptive redundant

information. A case study on NSS is provided in Appendix C.

More Comparison & Efficiency Analysis Beyond improving execution accuracy, JOLT-SQL also demonstrates superior training and inference efficiency compared to pipeline SFT methods. See Appendix A.1 for details.

5 Conclusion

In this paper, we introduced JOLT-SQL, an innovative SFT framework for Text-to-SQL. JOLT-SQL features a novel joint loss tuning method and incorporates confusion-aware noisy schema sampling to enhance model robustness. Experimental results compellingly demonstrate that JOLT-SQL achieves excellent SQL execution accuracy and exhibits clear advantages in efficiency. We believe this work offers new perspectives for developing more efficient and robust Text-to-SQL systems.

Limitations

Despite the promising results, this work has several limitations. Firstly, our method was validated on relatively small-scale LLMs (7B and 14B parameters). Due to constraints in hardware resources and time, we were unable to conduct experiments to verify its effectiveness on larger models (e.g., Qwen2.5-Coder-32B). Secondly, while our experiments demonstrate JOLT-SQL's excellent efficiency during both training and inference, the method involves sophisticated adjustments to attention masks and custom logic integrated into the training iterations, particularly for dynamic attention mask generation and the noisy schema sampling process. This can introduce a degree of complexity at the code implementation level.

Acknowledgments

This work is supported by the Key Program of Natural Science Foundation of China (Grant No. U23A20316).

References

Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, and 1 others. 2023. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*.

Hasan Alp Caferoğlu and Özgür Ulusoy. 2024. E-sql: Direct schema linking via question enrichment in text-to-sql. *arXiv preprint arXiv:2409.16751*.

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

Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, pages 2166–2187, Gyeongju, Republic of Korea. International Committee on Computational Linguistics.

Xiang Deng, Ahmed Hassan Awadallah, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-SQL. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 1337–1350, Online. Association for Computational Linguistics.

Tim Dettmers, Mike Lewis, Sam Shleifer, and Luke Zettlemoyer. 2022. 8-bit optimizers via block-wise quantization. *9th International Conference on Learning Representations, ICLR*.

Qingxiu Dong, Lei Li, Damai Dai, Ce Zheng, Jingyuan Ma, Rui Li, Heming Xia, Jingjing Xu, Zhiyong Wu, Baobao Chang, Xu Sun, Lei Li, and Zhifang Sui. 2024. A survey on in-context learning. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 1107–1128, Miami, Florida, USA. Association for Computational Linguistics.

Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Jinshu Lin, Dongfang Lou, and 1 others. 2023. C3: Zero-shot text-to-sql with chatgpt. *arXiv preprint arXiv:2307.07306*.

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-SQL models against synonym substitution. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2505–2515, Online. Association for Computational Linguistics.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 8926–8931, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024a. Text-to-sql empowered by large language models: A benchmark evaluation. *Proceedings of the VLDB Endowment*, 17(5):1132–1145.

Yingqi Gao, Yifu Liu, Xiaoxia Li, Xiaorong Shi, Yin Zhu, Yiming Wang, Shiqi Li, Wei Li, Yuntao Hong, Zhiling Luo, and 1 others. 2024b. Xiyan-sql: A multi-generator ensemble framework for text-to-sql. *arXiv preprint arXiv:2411.08599*.

Michael Glass, Mustafa Eyceoz, Dharmashankar Subramanian, Gaetano Rossiello, Long Vu, and Alfio Gliozzo. 2025. Extractive schema linking for text-to-sql. *arXiv preprint arXiv:2501.17174*.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2024. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv e-prints*, pages arXiv–2406.

Pin-Lun Hsu, Yun Dai, Vignesh Kothapalli, Qingquan Song, Shao Tang, Siyu Zhu, Steven Shimizu, Shivam Sahni, Haowen Ning, and Yanning Chen. 2024. Liger kernel: Efficient triton kernels for llm training. *arXiv preprint arXiv:2410.10989*.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, and 1 others. 2024. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*.

Aaron Hurst, Adam Lerer, Adam P Goucher, Adam Perelman, Aditya Ramesh, Aidan Clark, AJ Ostrow, Akila Welihinda, Alan Hayes, Alec Radford, and 1 others. 2024. Gpt-4o system card. *arXiv preprint arXiv:2410.21276*.

Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 2261–2273, Online. Association for Computational Linguistics.

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

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, and 1 others. 2023a. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36:42330–42357.

Zongxi Li, Xianming Li, Yuzhang Liu, Haoran Xie, Jing Li, Fu-lee Wang, Qing Li, and Xiaoqin Zhong. 2023b. Label supervised llama finetuning. *arXiv preprint arXiv:2310.01208*.

Haoyuan Ma, Yongliang Shen, Hengwei Liu, Wenqi Zhang, Haolei Xu, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Db-explore: Automated database exploration and instruction synthesis for text-to-sql. *arXiv preprint arXiv:2503.04959*.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Advances in Neural Information Processing Systems*, 36:36339–36348.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pages 8212–8220, Miami, Florida, USA. Association for Computational Linguistics.

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

Lei Sheng, Shuai-Shuai Xu, and Wei Xie. 2025. Base-sql: A powerful open source text-to-sql baseline approach. *arXiv preprint arXiv:2502.10739*.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2025. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2).

Ruoxi Sun, Sercan Ö Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and 1 others. 2023. Sql-palm: Improved large language model adaptation for text-to-sql (extended). *arXiv preprint arXiv:2306.00739*.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. Chess: Contextual harnessing for efficient sql synthesis. *arXiv preprint arXiv:2405.16755*.

Gemini Team, Petko Georgiev, Ving Ian Lei, Ryan Burnell, Libin Bai, Anmol Gulati, Garrett Tanzer, Damien Vincent, Zhufeng Pan, Shibo Wang, and 1 others. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv preprint arXiv:2403.05530*.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International*

Conference on Computational Linguistics, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, and 1 others. 2022. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837.

An Yang, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chengyuan Li, Dayiheng Liu, Fei Huang, Haoran Wei, and 1 others. 2024a. Qwen2. 5 technical report. *arXiv preprint arXiv:2412.15115*.

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

Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024c. Sql-to-schema enhances schema linking in text-to-sql. In *International Conference on Database and Expert Systems Applications*, pages 139–145. Springer.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921, Brussels, Belgium. Association for Computational Linguistics.

Appendix

A Further Analysis

A.1 Comparison with Pipeline Fine-tuning Strategies & Efficiency Analysis

To further demonstrate the advantages of JOLT-SQL, this section compares it with DTS-SQL (Pourreza and Rafiei, 2024), a typical pipeline fine-tuning method. DTS-SQL employs a two-stage pipeline: first, fine-tuning an LLM for generative schema linking, and then fine-tuning another LLM for SQL generation based on ground truth schema input (table-level). We re-evaluated DTS-SQL using experimental settings consistent with those for JOLT-SQL. To align with our experimental setup, the SQL generation fine-tuning phase for DTS-SQL was based on column-level ground truth schema input. We compared their training efficiency, inference speed, and task performance, as shown in Table 6.

In terms of training cost, the results show that JOLT-SQL requires 5 hours and 5 minutes to complete 3 epochs of training. This duration includes an approximate 8.5% overhead from enabling Confusion-aware NSS compared to its own baseline of 4 hours and 41 minutes (JOLT-SQL without Confusion-aware NSS), a time comparable to the standard SFT approach. Even so, JOLT-SQL is significantly more efficient than DTS-SQL. DTS-SQL, requiring independent fine-tuning for two stages, has a total training time of 7 hours and 10 minutes, which is 52.9% longer than the JOLT-SQL baseline (without Confusion-aware NSS), highlighting the efficiency of JOLT-SQL's single-stage joint optimization. This is because JOLT-SQL learns both tasks from a single input sequence, fundamentally halving the total number of forward/backward passes compared to two-stage methods that process separate data for each task.

In terms of inference speed, JOLT-SQL's discriminative schema linking shows a significant advantage, averaging 0.11 seconds, which is much faster than DTS-SQL's generative linking at 0.57 seconds—the latter being more than five times slower. DTS-SQL's schema linking is more time-consuming mainly because it requires autoregressively generating relevant table and column names token by token during inference. In contrast, JOLT-SQL's discriminative method requires only a single full forward pass to determine the relevance of all schema candidates. Thanks to its efficient schema linking, JOLT-SQL's average end-to-end inference time is only 0.88 seconds, whereas DTS-SQL takes 1.34 seconds (approximately 52.3% slower), with the difference primarily stemming from the schema linking stage.

In terms of Execution Accuracy (EX), JOLT-SQL (87.0%) outperforms DTS-SQL (84.8%) on Spider Dev. This difference may stem from the characteristics of their schema linking approaches and the robustness of their SQL generation stages, as further elaborated by Table 7.

Table 7 compares the schema linking metrics. DTS-SQL's generative linking achieves high precision 93.67%, with a recall of 94.15%. However, as discussed in the main text, JOLT-SQL's discriminative method can flexibly use a low decision threshold of 0.05 to prioritize recall, achieving a recall rate as high as 98.12%. In Text-to-SQL tasks, high recall is crucial: omitting necessary items often leads to SQL failure, whereas a few redundant items have less impact if the SQL model

Methods	Total Training Time↓	Avg. Schema Linking Inference Time↓	Avg. SQL Generation Inference Time↓	Avg. End-to-End Inference Time↓	Spider Dev EX
Qwen2.5-Coder-7B + SFT	4h38min	-	0.94s	0.94s	82.7
DTS-SQL(Pourreza and Rafiei, 2024) + Qwen2.5-Coder-7B	7h10min(+52.9%)	0.57s(+418.2%)	0.77s	1.34s(+54.0%)	84.8
JOLT-SQL(**Ours**) + Qwen2.5-Coder-7B	5h5min(+8.5%)	**0.11s**	**0.77s**	**0.88s**	**87.0**
w/o Confusion-aware NSS(**Baseline**)	**4h41min**	**0.11s**	**0.76s**	**0.87s**	86.1

Table 6: Comparison results with the two-stage pipeline fine-tuning method DTS-SQL and a standard SFT approach (Qwen2.5-Coder-7B + SFT) are presented, all evaluated on a single NVIDIA A30 GPU. KV cache was enabled for all inference processes. We use JOLT-SQL without Confusion-aware NSS as the baseline for comparing training and inference times on the Spider Dev set. Training time is the total for 3 epochs. Inference time is the average per instance with Batchsize=1. For context, the cost for a single NVIDIA A30 instance was approximately $0.18 per hour at the time of the experiments. **Additional Notes:** • The slightly longer SQL generation inference time for the standard SFT approach (Qwen2.5-Coder-7B + SFT) is due to its processing of the full schema, which results in longer input sequences. • It's worth noting the distinction in forward pass latency between training and inference: our use of LoRA means adapters introduce some overhead during the training's forward pass, but for inference, these LoRA weights are merged into the base model, eliminating this adapter-induced latency.

Methods	Spider Dev			
	P	R	ROC	PR
DTS-SQL(Pourreza and Rafiei, 2024) + Qwen2.5-Coder-7B	**93.67**	94.15	N/A	N/A
JOLT-SQL(**Ours**) + Qwen2.5-Coder-7B	88.09	**98.12**	**99.86**	**98.70**

Table 7: Comparison of schema linking results between DTS-SQL's method (i.e., generative schema linking based on LLM SFT) and JOLT-SQL's discriminative method. **P** and **R** represent column Precision and column Recall, respectively. Notably, since generative LLMs output discrete schema linking results directly, probability information is not readily available, making standard ROC AUC and PR AUC metrics inapplicable.

Methods	Spider-Syn	Spider-Realistic	Spider-DK	KaggleDBQA
JOLT-SQL + Qwen2.5-Coder-7B	77.2	83.0	75.9	47.0
JOLT-SQL + Qwen2.5-Coder-14B	79.6	84.1	79.1	52.4

Table 8: Execution Accuracy of JOLT-SQL on Spider variants and KaggleDBQA. The models were fine-tuned only on the original Spider training set.

is robust. The insufficient recall of DTS-SQL's schema linking might be one of the core reasons for its lower EX. Additionally, its SQL generation model is fine-tuned only on ground-truth relevant columns, lacking adaptability to imperfect inputs from actual schema linking.

The results again demonstrate that JOLT-SQL's joint loss tuning strategy is significantly superior to DTS-SQL's two-stage method in both training and inference efficiency. Its combination of discriminative schema linking and Confusion-aware NSS achieves better recall and model robustness, ultimately improving SQL EX.

A.2 Cross-Dataset Generalization

Table 8 presents JOLT-SQL's robustness and generalization capabilities across various challenging datasets. We evaluated its performance on Spider-Syn (Gan et al., 2021a), Spider-Realistic (Deng et al., 2021), Spider-DK (Gan et al., 2021b), and the cross-domain KaggleDBQA dataset (Lee et al., 2021). The results indicate that JOLT-SQL maintains strong performance. Furthermore, it demonstrates solid generalization to entirely unseen databases and query types in KaggleDBQA.

A.3 Performance in Low-Resource Settings

Training data used	Spider Dev EX	BIRD Dev EX
5%	81.9	54.0
10%	83.8	55.3
25%	85.4	58.5
50%	86.4	59.7
100%	**87.0**	**60.4**

Table 9: Ablation results for JOLT-SQL under reduced data conditions (based on Qwen2.5-Coder-7B).

Table 9 evaluates JOLT-SQL's performance under low-resource training conditions. We fine-tuned the Qwen2.5-Coder-7B model using vari-

ous proportions of the Spider and BIRD training set. The results indicate that the model's execution accuracy degrades gracefully as training data decreases, without a sharp drop. Even with only 5% or 10% of the data, JOLT-SQL still achieves acceptable performance, demonstrating its robustness and practical applicability in scenarios with limited annotated data.

B More Details

B.1 Training Data Construction

We use SQLGlot[1] to process SQL queries. By parsing SQL query statements to construct their Abstract Syntax Trees (ASTs) and leveraging SQLGlot's scope analysis feature, we can accurately identify each column referenced in the query and its originating data table, while also resolving table aliases. This ultimately allows us to extract ground truth schema linking pairs in the `table.column` format.

Figure 5 shows an example of the actual data we use. The `link` field contains the extracted ground-truth schema linking pairs. Based on this, we generate the final ground truth labels for schema linking (i.e., the `label` field).

We use `schema_element_token_spans` field to record the positions of all schema elements. This is a nested dictionary structure that records the precise token spans for various elements within the schema. In addition to column definitions, it includes keys such as `fk` (foreign key definition span), `header` (table header definition span), `pk` (primary key definition span), and `footer` (table definition end span), which are used to accurately locate different components of the schema.

B.2 Implementation Details

All our experiments were conducted on NVIDIA A30 GPUs with 24GB VRAM. The 7B models were trained on a single A30 GPU, while the 14B models used 2xA30 GPUs. The frameworks we used were PyTorch 2.5.1 and HuggingFace Transformers 4.51.3.

We employ LoRA (Hu et al., 2022) as the training method to reduce VRAM requirements. The target modules for LoRA are all linear layers in the LLM decoder. In addition to the LoRA trainable parameters, the linear layer weight W (Equation 1) is also part of the trainable parameters.

[1]https://github.com/tobymao/sqlglot


```
"instruction": "'Please generate SQLite query based on the following question and schema.
### Question:
For each fourth-grade classroom, show the classroom number and the total number of students using it.

### Schema:
<schema>
CREATE TABLE list (  LastName TEXT, Example: ['AMY', 'AREHART']<|endoftext|>
 FirstName TEXT, Example: [' PATRINA', ' VERTIE']<|endoftext|>
 Grade INTEGER, Example: [2, 6]<|endoftext|>
 Classroom INTEGER, Example: [101, 112]<|endoftext|>
 PRIMARY KEY (LastName, FirstName)
);

CREATE TABLE teachers (  LastName TEXT, Example: ['COVIN', 'KAWA']<|endoftext|>
 FirstName TEXT, Example: [' JEROME', ' GORDON']<|endoftext|>
 Classroom INTEGER, Example: [101, 102]<|endoftext|>
 PRIMARY KEY (LastName, FirstName)
);

</schema>'",
"output": "SELECT classroom ,  count(*) FROM list WHERE grade  =  \"4\" GROUP BY classroom",
"db_id": "student_1",
"link": ["list.grade", "list.classroom"],
"label": [0, 0, 1, 1, 0, 0, 0],
"schema_element_token_spans": {
 "tables": {
  "list": {
   "columns": {
    "LastName": [4, 20], "FirstName": [20, 38],
    "Grade": [38, 52], "Classroom": [52, 70]
   },
   "fk": [], "header": [0, 4], "pk": [70, 78], "footer": [78, 80]
  },
  "teachers": {
   "columns": {
    "LastName": [84, 101], "FirstName": [101, 119],
    "Classroom": [119, 137]
   },
   "fk": [], "header": [80, 84], "pk": [137, 146], "footer": [146, 148]
  }
 }
}
```


Figure 5: An example of actual training data.

Hyperparameters	
Epoch	3(Spider),2(BIRD)
Batchsize	1
Gradient Accumulation	6
Learning Rate	1.8e-5
Weight Decay	1e-4
Max Grad Norm	1.0
Truncation Max Length	4096
LoRA Rank	64
LoRA Alpha	512
LoRA Dropout	0.08

Table 10: Hyperparameters used for training.

We consistently used the AdamW8bit optimizer (Dettmers et al., 2022), a cosine annealing learning rate scheduler, and enabled gradient checkpointing. We also utilized the Liger-Kernel (Hsu et al., 2024) for further training efficiency optimization and enabled bfloat16 mixed-precision training. Other training hyperparameters are provided in Table 10. For SQL generation during inference, we employed

greedy decoding.

C Case Study of Noisy Schema Sampling

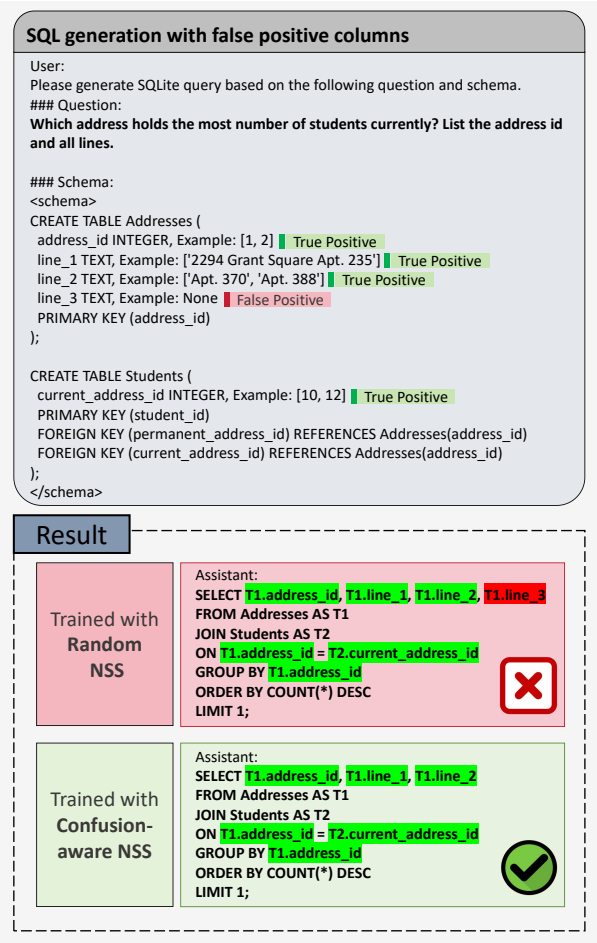


Figure 6: A comparative case study of SQL generation with Random NSS versus Confusion-aware NSS when handling columns with False Positives. Green indicates True Positive columns, and red indicates False Positive columns. The model trained with Confusion-aware NSS (our method) correctly ignores the False Positive column `line_3`, whereas Random NSS incorrectly includes it in the query.

To more intuitively demonstrate the robustness differences of various Noisy Schema Sampling (NSS) strategies in handling False Positive (FP) columns introduced by schema linking in SQL generation tasks, we conduct a case study, as illustrated in Figure 6. In this case, the user's question is: *"Which address holds the most number of students currently? List the address id and all lines."*

The provided database schema includes an *Addresses* table and a `Students` table. In the `Addresses` table, `address_id`, `line_1`, and `line_2` are True Positive (TP) columns relevant to the question, while `line_3` is a False Positive (FP) column. Its example value is `None`, indicating it is an empty column with no data, which was mistakenly identified as relevant during the schema linking.

We compare the model performance under two NSS training strategies: (1) **Model trained with Random NSS**: The generated SQL query incorrectly includes the FP column `line_3`. This indicates that despite being exposed to random noise during training, the model failed to adequately learn to ignore such misleading FP columns, especially when the column `line_3` has some textual similarity to "*all lines*" in the question. (2) **Model trained with Confusion-aware NSS**: The generated SQL query correctly ignores the FP column `line_3` and selects only the truly relevant columns. This demonstrates that Confusion-aware NSS, by enabling the model to focus on learning its own 'easily confused' noisy patterns during training, can more effectively enhance the model's robustness to schema linking errors.