Learning SQL Like a Human: Structure-Aware Curriculum Learning for Text-to-SQL Generation

Anonymous ACL submission

Abstract

The Text-to-SQL capabilities of large language allow users to interact with databases using natural language. While current models struggle with handling complex queries, especially involving multi-table joins and reasoning. To address this gap, we propose to construct a model, namely SAC-SQL, with synthetic training samples followed by a structure-aware curriculum learning framework for enhancing SQL generation. Our approach begins with a supervised fine-tuning (SFT) stage, where we train open-source models on a synthetically constructed, cross-domain SQL dataset with diverse structural patterns. Moreover, we introduce a unified structure difficulty scoring function to partition the training samples into non-overlapping curriculum phases, guiding the model progressively learning from simpler to more complex SQL structures. Extensive experiments are conducted and the results show that SAC-SQL achieves better results than the baselines, and significantly narrows the performance gap between open-source and close-source models on Spider and Bird benchmarks.

1 Introduction

Text-to-SQL generation task aims to automatically translate natural language questions into executable SQL queries, allowing users to interact with databases using plain language(Li et al., 2023c). This task reduces the technical barrier for data access and plays a key role in natural language interfaces to structured data (Qin et al., 2022a).

Several methods have been proposed to achieve the goal. Some methods adopt prompt engineering to solve this task. Besides, some method adopt agent-based methods, and divide the task into multiple subtasks. Besides, some methods propose to collect many related data, and adopt instruction tuning to train models. Among them, closed-source GPT-4(OpenAI et al., 2024) and open-source CodeLLaMA(Rozière et al., 2024) have led to remarkable progress on Text-to-SQL generation task,

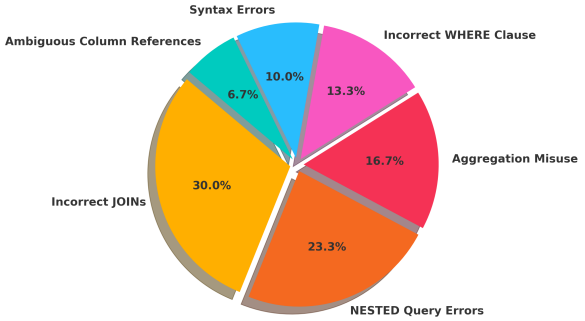

Figure 1: Distribution of error types for complex SQL queries (total errors = 60).

However, existing methods still struggle with handling complex queries that involve multi-table joins and reasoning. According to our statistics, CodeLLaMA achieves an execution accuracy of 86% for easy queries compared to only 40% for complex queries[1]. Furthermore, we performed an error analysis on the 30 incorrectly predicted complex samples (totaling 60 individual errors, with some examples containing multiple issues). As shown in Figure 1, the distribution of error types revealed that *Incorrect JOINs* accounted for the largest share (30%), followed by *NESTED Query Errors* (23.3%), *Aggregation Misuse* (16.7%), *Incorrect WHERE Clause* (13.3%), *Syntax Errors* (10%), and *Ambiguous Column References* (6.7%). These findings clearly indicate that these LLMs struggle particularly with structurally complex queries such as JOIN operations and nested queries.

To overcome this challenge, in this paper, we proposed a novel model, SAC-SQL, which adopts synthetic training samples followed by a structure-aware curriculum learning framework for enhancing SQL generation. Our model builds upon two

[1]We randomly sampled 100 examples from existing Text-to-SQL datasets and categorized them into *easy* and *complex* queries based on their structural complexity.

Dataset	#Examples	#Databases	Avg Token	Avg JOIN	NESTED Ratio
Spider	7000	140	37.3	0.54	5.6%
Bird	9428	69	64.0	1.02	12.3%
Ours	12134	674	56.7	1.26	**18.7%**

Table 1: Comparison of dataset properties across Spider, Bird, and our synthetic dataset.

key insights: First, text-to-SQL difficulty is governed by query structure, and current models fail to generalize well to complex structures when trained in a uniform manner(Shi et al., 2024). Second, high-quality real-world Text-to-SQL datasets are scarce and difficult to obtain, limiting the diversity and complexity of training examples. Thus, we propose to construct a high-quality, synthetically generated SQL dataset that covers a wide variety of query structures and schema domains.

Besides, we propose a two-stage training strategy to train the models. At the first stage, the synthesized dataset is used to supervise fine-tune open-source language models to provide them with essential SQL syntax and schema grounding. By explicitly controlling the composition of SQL structures during generation, we ensure that more examples contain low-frequency but high-difficulty patterns. This helps alleviate the long-tail distribution problem and enhances the model's ability to generalize to rare but semantically important query types(Yang et al., 2024).

At the second stage, we introduce a structure-aware curriculum learning framework to further enhance the model's ability to handle structurally complex queries. We design a unified function based on structural components, interaction patterns, syntactic tree depth, and generation uncertainty (via normalized NLL). Training samples are ranked and divided into non-overlapping phases of increasing difficulty, allowing the model to learn SQL structure in a staged, cognitively aligned way.

Combining these two-stage processes, we develop SAC-SQL, a curriculum-enhanced model trained atop an SFT-initialized open-source LLM. Experiments on Spider and Bird benchmarks show that SAC-SQL achieves better results than existing methods, and surpasses existing open-source baselines. Besides, it outperforms GPT-4 and its derivatives in executing complex SQL queries, highlighting the effectiveness of combining data design and curriculum scheduling in improving SQL compositionality and reasoning.

The contribution of this work is as follows,

- We propose a novel model SAC-SQL trained on synthetic samples covering a wide variety of query structures and schema domains.
- We propose a two-stage training pipeline. Considering the characteristics of the synthetic sample, we design a structure-aware curriculum learning with difficulty measure at the second stage.
- Extensive experiments prove the superiority of SAC-SQL over state-of-the-art baselines.

2 Related Work

LLM-based Text-to-SQL Recent advances in large language models (LLMs) have led to a surge of approaches designed to improve Text-to-SQL performance by leveraging pretraining, schema understanding, and task-specific optimization. Among them, Knowledge-to-SQL(Hong et al., 2024) enhances database-schema alignment by combining supervised fine-tuning with Direct Preference Optimization (DPO)(Rafailov et al., 2024), effectively improving execution accuracy. SGU-SQL(Zhang et al., 2024a) adopts graph-structured representations and grammar-aware parsing to strengthen query-schema linkage and compositional reasoning. CLLMs(Kou et al., 2024) introduces a consistency loss as an implicit regularization mechanism to stabilize convergence and improve model adaptability. StructLM(Zhuang et al., 2024) proposes a structured learning framework that integrates code-level pretraining with cued tuning and constrained parameter adaptation via regularization techniques. MCS-SQL(Lee et al., 2024) combines schema linking, multi-path SQL generation, and candidate reranking; it employs multiple prompts to sample a wide reasoning space and selects the best SQL candidate through execution-aware scoring functions.

Curriculum Learning Curriculum Learning (CL)(Dai et al., 2021) is a training paradigm that organizes examples in an easy-to-hard order, inspired by the way humans progressively acquire

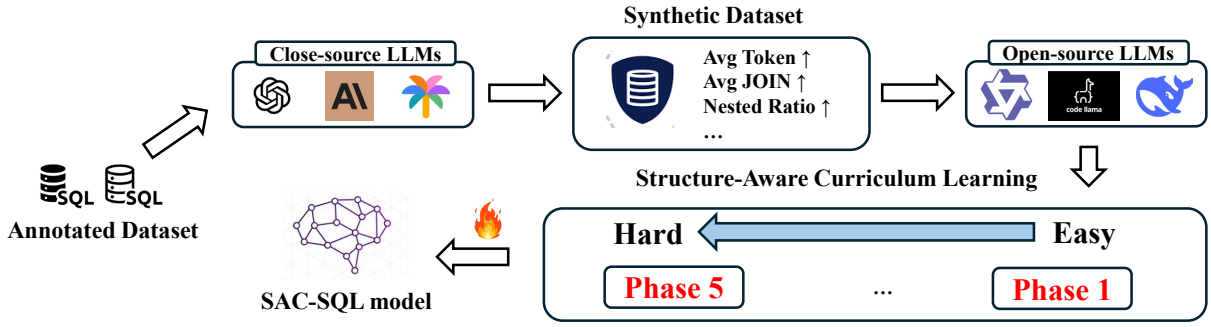

Figure 2: An overview of SAC-SQL.

knowledge. Instead of randomly sampling from the full dataset, CL gradually increases the difficulty of training samples to align with the learner's growing capacity(Bengio et al., 2009). This paradigm has been widely applied in domains such as machine translation, reading comprehension, and dialog modeling, where task complexity and compositionality vary across samples.(Wang et al., 2021) In Text-to-SQL generation, queries differ significantly in their structural complexity—ranging from single-table selections to deeply nested multi-join queries. Models trained on such data in a uniform or random fashion often fail to generalize to the long tail of complex SQL structures. For example, CL can be applied to the neural machine translation task, which significantly improves the performance of the model under low resource conditions(Zhang et al., 2018; Dou et al., 2022). By introducing curriculum learning into Text-to-SQL, we aim to shape the model's training trajectory so that it first masters basic constructs, then gradually progresses toward more compositional logic.

3 Methodology

Our training process consists of two major stages: a supervised fine-tuning (SFT) phase and a subsequent structure-aware curriculum learning phase. The overall goal is to improve the ability of large language models to handle structurally complex SQL queries by gradually exposing them to increasingly difficult training examples. Figure 2 illustrates the full pipeline.

3.1 Synthetic Data Generation

We leverage GPT-4 and Cladue-2 to synthesize a high-quality, cross-domain Text-to-SQL dataset that exhibits greater structural diversity compared to existing benchmarks. Our synthetic dataset includes a richer distribution of SQL functions, with increased frequency and variation in constructs such as `JOIN`, `GROUP BY`, `HAVING`, `WINDOW`, and `UNION`, addressing the structural sparsity observed in datasets like Spider and Bird. Furthermore, unlike these benchmarks which suffer from domain concentration, our dataset is drawn from over 500 distinct databases. Table 1 provides a comparative overview of the structural characteristics of the Spider, Bird, and our synthetic dataset.

3.2 Structure-aware difficulty function

To guide the model's training trajectory in a cognitively-aligned fashion, we define a unified structure-aware difficulty function to evaluate the complexity of each SQL example x. This function integrates both the inherent structural difficulty of the SQL query and the model's uncertainty in generating it. The resulting score serves as the basis for curriculum phase partitioning. Each training sample is assigned a score $D(x)$, which captures both the SQL's compositional complexity and the model's generation difficulty,

$$D(x) = \alpha \cdot S(x) + \beta \cdot I(x) + \gamma \cdot T(x) + \delta \cdot L(x) \quad (1)$$

where:

- $S(x)$ captures the base complexity of SQL components present,
- $I(x)$ accounts for interaction between components (co-occurrence patterns),
- $T(x)$ measures the syntactic depth of the SQL expression tree, and
- $L(x)$ reflects model-specific generation uncertainty based on token-level loss.

Each coefficient α, β, γ, δ is a tunable hyperparameter that controls the relative influence of the four difficulty dimensions.

SQL Function	Weight
SELECT	1.0
WHERE	0.5
JOIN	1.5
GROUP BY	1.0
HAVING	1.2
ORDER BY	0.8
LIMIT	0.5
NESTED SELECT	1.5
WINDOW / RANK	2.0
UNION / INTERSECT	2.0

Table 2: Structure component weights for $S(x)$.

To design the base structure weights in $S(x)$ and interaction scores in $I(x)$, we start with linguistically and semantically informed heuristics. Each weight reflects the compositional and logical burden imposed by a SQL component. In Section 4.5, we further explore the sensitivity of these weights and validate their impact on model performance.

Base Structure Score $S(x)$

We define a structure component dictionary assigning difficulty weights to common SQL constructs, To quantify the structural complexity of each SQL query, we define a component-wise scoring function $S(x)$ that assigns weights to different SQL operations based on their syntactic and semantic difficulty. Table 2 lists the weights used for each function, where more compositional or computationally intensive operations (e.g., WINDOW, NESTED SELECT, UNION) receive higher scores.

Each SQL query is parsed to determine which structures are present. $S(x)$ is the sum of the corresponding weights of those constructs.

Interaction Score $I(x)$

Structural difficulty often increases when multiple components co-occur. For example, JOIN + HAVING and GROUP BY + WINDOW represent compounded reasoning. Beyond individual SQL functions, we also account for interaction complexity by modeling common co-occurring component pairs. These interactions often increase the compositional difficulty of a query. For example, queries involving both NESTED SELECT and HAVING clauses tend to exhibit deeper logical nesting and filtering logic. As shown in Table 3, we assign interaction scores in $I(x)$ to reflect the added difficulty introduced by such structural combinations.

Component Pair	Score
JOIN + GROUP BY	0.8
JOIN + HAVING	1.2
NESTED + HAVING	1.5
GROUP BY + WINDOW	1.0

Table 3: Interaction component pairs for $I(x)$.

Syntactic Depth Score $T(x)$

Using the SQL's parsed abstract syntax tree (AST), we define $T(x)$ as a linear combination of:

- Tree depth d: number of nested clauses or expression layers,
- Number of subqueries q: e.g., the count of NESTED SELECT statements.

$$T(x) = \lambda_1 \cdot d + \lambda_2 \cdot q \tag{2}$$

where typical values are $\lambda_1 = 0.5$, $\lambda_2 = 0.8$. This term captures the compositional complexity of the SQL's logical form.

Model Uncertainty Score $L(x)$

We define $L(x)$ as the normalized token-level negative log-likelihood (NLL) of the model when generating SQL example x:

$$L(x) = \frac{\text{NLL}(x) - \mu}{\sigma} \tag{3}$$

where μ and σ are the mean and standard deviation of NLL values across the training set. This score reflects model-side uncertainty, with larger $L(x)$ indicating greater difficulty in generation. In our experiments, we set $\alpha = \beta = \gamma = 1.0$ and $\delta = 0.5$ to moderately incorporate model uncertainty. These coefficients can be tuned to suit different curriculum learning needs .

3.3 Curriculum Phase Scheduling

Once each training sample x is assigned a structure-aware difficulty score $D(x)$, we partition the dataset into a sequence of non-overlapping curriculum phases, each corresponding to a difficulty band. This phased scheduling ensures that the model is exposed to easier examples first and gradually progresses to more complex samples as training proceeds. We discretize the continuous score range of $D(x)$ into K buckets using fixed thresholds $\{\tau_1, \tau_2, \ldots, \tau_{K-1}\}$ such that each sample belongs to exactly one phase. Specifically, we define: Phase 1 as $D(x) \in [0, \tau_1)$, Phase 2 as $D(x) \in [\tau_1, \tau_2)$, ..., and Phase K as $D(x) \geq \tau_{K-1}$.

In our implementation, we use $K = 4$ curriculum phases with the following thresholds: $[0, 3)$, $[3, 5.5)$, $[5.5, 7.5)$, and $[7.5, +\infty)$. These bands correspond to an increasing level of SQL complexity, where Phase 1 includes simple single-table queries with basic `SELECT`, `WHERE`, or `LIMIT` clauses; Phase 2 introduces `JOIN` and simple aggregation; Phase 3 includes multi-clause logic involving `GROUP BY`, `HAVING`, and moderate nesting; and Phase 4 contains highly complex queries featuring `NESTED SELECT`, `WINDOW`, or multiple structural interactions. The partitioning is performed using `np.digitize()` or a similar binning operation to ensure that each training instance is uniquely assigned to one phase.

To implement the structured training schedule, we denote $D_1, D_2, \ldots, D_K$ as the subsets of training samples assigned to each phase. Training is carried out in a sequential phase-wise manner: in epochs 1 to T_1, only D_1 is used; in epochs T_1 to T_2, $D_1 \cup D_2$ is used; and so on. In the final stage of training, the full dataset $\bigcup_{k=1}^{K} D_k$ is employed. This incremental exposure allows the model to master fundamental SQL generation patterns early, before being exposed to increasingly complex queries, resulting in more stable and effective learning. Additionally, we experiment with phase blending and loss re-weighting strategies, where earlier-phase samples are retained in later stages but assigned reduced loss weights, helping preserve their reinforcement signal without risking overfitting.

Compared to uniform sampling or randomly ordered curricula, our structured scheduling framework offers three key advantages: it ensures stable convergence by preventing premature exposure to outlier queries, it enhances structure-specific generalization—particularly on complex SQL constructs such as `JOIN`, `NESTED SELECT`, and `HAVING`—and it provides interpretable control over training dynamics via the design of curriculum thresholds.

4 Experiments

4.1 Benchmark and Metric

Benchmarks We evaluate the performance of SAC-SQL on two widely used Text-to-SQL benchmarks—Spider and Bird. Spider(Yu et al., 2019) is a cross-domain Text-to-SQL dataset featuring over 200 databases from 138 domains (e.g., education, science), with an average of 5.1 tables per database. Notably, the training and test sets contain disjoint databases to evaluate generalization.

BIRD(Li et al., 2023c) is a benchmark designed to bridge academic research and real-world applications by emphasizing grammatical clarity, ambiguity, specificity, and schema alignment. It spans 37+ domains (e.g., healthcare, hockey, education) and introduces challenges such as noisy database contents, the need for external knowledge, and query efficiency over large databases.

Evaluation Metrics For evaluation, we adopt different metrics tailored to each dataset's structure and objectives. On the Spider benchmark, we follow standard practice and report both Execution Accuracy (EX)(Yu et al., 2019) and Test Suite Accuracy (TS)(Zhong et al., 2020). EX measures whether the generated SQL produces the correct execution result when run on the database, while TS further accounts for a broader set of execution behaviors by evaluating correctness across multiple input-output cases per query.

For the Bird dataset, we rely solely on execution accuracy (EX). Although Bird provides its own official evaluation metrics, they are tightly coupled with dataset-specific constraints and scoring rules that do not generalize well to our open-source models or synthetic setups. Therefore, for consistency and interpretability, we evaluate Bird results exclusively using EX, which directly reflects semantic correctness and aligns with our primary focus on structural generalization.

4.2 Compared Methods

In order to comprehensively assess the performance of our proposed SAC-SQL model, we compare it against a broad range of baselines drawn from both close-source and open-source model families. To ensure fair and interpretable comparison, we categorize these baselines into four groups: closed-source LLMs, prompt-based methods, fine-tuning approaches, and open-source LLMs.

The first group, closed-source LLMs, includes proprietary systems such as GPT-4, PaLM-2(Anil et al., 2023), Claude-2(Anthropic, 2023), and ChatGPT. These models are known for their strong performance in natural language understanding and reasoning, but are often inaccessible for fine-tuning or domain-specific adaptation. Their inclusion provides an upper-bound reference for open-source model performance, especially in zero-shot or few-shot settings(Kim et al., 2020).

The second group consists of prompting methods based on in-context learning, such as few-shot

Methods		Spider			Bird	
		Dev-EX	Dev-TS	Test	Dev	Test
Closed Source LLMs	GPT-4	72.9	64.9	-	49.2	54.9
	PaLM-2	-	-	-	27.4	33.1
	Claude-2	-	-	-	42.7	33.1
	ChatGPT	72.3	-	-	36.6	40.1
Prompting Methods	Few-shot GPT-4	76.8	67.4	-	-	-
	Few-shot SQL-PaLM	82.7	77.3	-	-	-
	DIN-SQL + GPT-4	82.8	74.2	85.3	50.7	55.9
	ACT-SQL + GPT-4	82.9	74.5	-	-	-
	DAIL-SQL + GPT-4	83.5	76.2	86.6	54.8	57.4
Fine-tuning Methods	T5-3B + PICARD	79.3	69.4	75.1	-	-
	RASAT + PICARD	80.5	70.3	75.5	-	-
	RESDSQL-3B + NatSQL	84.1	73.5	79.9	-	-
	Graphix-T5-3B + PICARD	81.0	75.0	-	-	-
	Fine-tuned SQL-PaLM	82.8	78.2	-	-	-
Open Source LLMs	Llama2-7B	28.0	23.8	-	7.1	-
	Llama2-13B	36.9	34.9	-	11.3	-
	LLaMA2-13B-Chat	49.6	45.5	-	14.2	-
	DeepSeek-Coder-1.3B	59.3	53.2	-	22.0	-
	CodeLLaMA-7B	61.1	52.3	-	22.5	-
	CodeLLaMA-13B	61.7	53.5	-	22.9	-
	CodeLLaMA-7B-Instruct	63.4	54.2	-	23.0	-
	DeepSeek-Coder-1.3B-Instruct	53.2	48.7	-	24.1	-
	Qwen-7B	63.6	54.5	-	26.1	-
Ours	SAC-SQL-7B	81.7	80.1	81.9	50.8	57.3
	SAC-SQL-13B	**83.2**	**81.5**	**84.1**	**53.9**	**59.7**

Table 4: Comparison of SAC-SQL with baseline methods on Spider and Bird datasets. SAC-SQL demonstrates strong performance across all settings, particularly on structurally complex queries.

GPT-4, SQL-PaLM(Sun et al., 2024), and several recent techniques that combine powerful LLMs with specialized SQL-oriented prompts (e.g., DIN-SQL(Pourreza and Rafiei, 2023), ACT-SQL(Zhang et al., 2023), and DAIL-SQL(Gao et al., 2023)). These approaches do not involve parameter updates but rely on advanced prompting strategies to induce SQL generation capabilities. They are typically more flexible and require no task-specific training, but may underperform on structurally difficult or domain-shifted queries.

The third group includes fine-tuning methods, where large models such as T5(Raffel et al., 2023), RESDSQL(Li et al., 2023a), or Graphix-T5(Li et al., 2023b) are trained end-to-end on Text-to-SQL datasets. Many of these methods are enhanced with decoding constraints (e.g., PICARD(Scholak et al., 2021)) or logical normalization (e.g., NatSQL(Gan et al., 2021)). They represent the strongest supervised learning baselines and are directly comparable to SAC-SQL in terms of training strategy and evaluation protocol(Qin et al., 2023).

Finally, we include a set of open-source LLMs without task-specific fine-tuning, including models from the LLaMA, CodeLLaMA, Qwen(Bai et al., 2023), and DeepSeek(Guo et al., 2024) families. These models are used in either base or instruct-tuned form, providing a lower bound for performance and helping isolate the gains achieved by synthetic data, supervised fine-tuning.

This wide-ranging comparison allows us to position SAC-SQL not only against state-of-the-art supervised models, but also in relation to widely-used systems and emerging open-source baselines.

4.3 Implementations Details

Our experiments are conducted with open-source decoder-style language models based on the CodeLLaMA architecture, with both 7B and 13B parameter variants. The models are trained with mixed-precision on 2×NVIDIA A100 GPUs, each with 80GB of memory. Model training and evaluation are orchestrated through the deepspeed and accelerate backends to ensure efficient memory scaling.

We fine-tune each model using the AdamW optimizer with a linear learning rate schedule, warm-up ratio of 0.03, and a base learning rate of 1e-5. During SFT, models are trained on our synthetic dataset for 6 epochs with a batch size of 128 and context length of 2048. Curriculum training proceeds in four non-overlapping phases defined by the structure-aware score D(x), with phase transitions scheduled every 2 epochs. In each phase, only the corresponding subset of data is exposed to the model, following a strict easy-to-hard progression. We apply no early stopping, and select the best

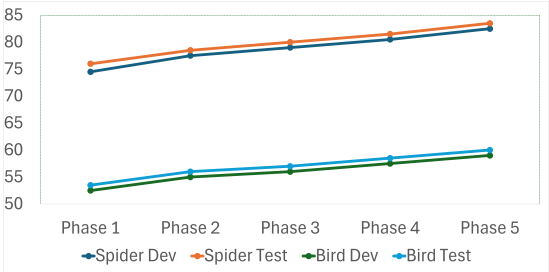

Figure 3: Execution accuracy (EX) on Spider and Bird datasets as more curriculum phases are introduced.

checkpoint based on dev execution accuracy.

During training, SQL queries are linearized using a deterministic schema serialization format, and both Spider and Bird are preprocessed to unify case, keyword spacing, and quoting conventions. Schema information is appended to the input prompt for each query using a template consistent with prior work. No execution feedback or reinforcement learning signals are used—our model is trained entirely via supervised objectives. Final evaluation is conducted using the official Spider toolkit and an adapted Bird execution evaluator, using execution accuracy (EX) as the primary metric.

4.4 Overall Performance

Table 4 presents a comprehensive comparison of SAC-SQL with a wide range of baselines across the Spider and Bird datasets. Our model achieves competitive or superior results on both benchmarks, outperforming previous open-source models and matching or exceeding the performance of several closed-source methods.

On the Spider dataset, SAC-SQL-13B reaches 83.2% execution accuracy (EX) and 81.5% test suite accuracy (TS) on the dev set, and 84.1% EX on the test set—significantly surpassing the supervised fine-tuning (SFT) baseline and all other open-source LLMs. Compared to strong supervised methods such as Graphix-T5 or RESDSQL, SAC-SQL exhibits more stable performance without relying on additional decoding constraints like PICARD. While prompt-based methods (e.g., DAIL-SQL + GPT-4) perform strongly in zero-shot settings, they often lack consistency on structurally difficult queries, especially when schema composition is dense. SAC-SQL, in contrast, benefits from progressive exposure to complex queries during training, allowing it to generalize more reliably across different SQL compositions.

On the Bird benchmark, which features more natural and varied linguistic inputs, SAC-SQL-13B achieves 59.7% test execution accuracy, outperforming GPT-4 (54.9%) and all other open-source and prompting baselines. Notably, this result is obtained using execution accuracy (EX) alone, rather than relying on the dataset's official scoring rules, which are incompatible with standard open-source model usage. The large margin between SAC-SQL and the SFT baseline (which achieves only 50.5% EX) underscores the impact of curriculum learning on structural transfer and long-tail generalization. To investigate the effectiveness of our curriculum learning framework, we measure how execution accuracy evolves as training data gradually incorporates more structurally complex SQL samples. As shown in Figure 3, model performance on both Spider and Bird benchmarks improves steadily with each additional phase. This progression illustrates the cognitive benefit of structure-aware scheduling—early exposure to simple patterns like `SELECT` and `WHERE` forms a syntactic foundation, which facilitates later generalization to more difficult constructs such as `JOIN`, `GROUP BY`, and `NESTED SELECT`. Notably, the final performance after Phase 5 outperforms all previous checkpoints, confirming that phased curriculum training yields stronger and more stable convergence than flat SFT.

Overall, the results demonstrate that SAC-SQL not only closes the gap between open-source and closed-source models, but in many cases, surpasses proprietary methods on structurally rich and semantically complex SQL tasks. The performance gains are particularly evident in scenarios involving nested logic, rare aggregation patterns, and multi-clause queries—areas that standard fine-tuning tends to underperform without structural guidance.

4.5 Weight Sensitivity Analysis

To assess the impact of different structural weighting configurations in the base difficulty score $S(x)$, we conduct an ablation study by varying the component weights used in our curriculum scheduler. Table 5 and Figure 4 summarize the effect of altering weights for challenging SQL components such as `JOIN`, `NESTED SELECT`, and `WINDOW`.

The results show that the heuristic configuration yields the best overall performance. Reducing the weight of `JOIN` leads to a 1.7% drop in Spider accuracy, while removing the penalty for `NESTED SELECT` results in degraded handling of long-tail queries in Bird. A uniform weight setting underperforms due to lack of structural contrast. These findings validate the importance of accurately re-

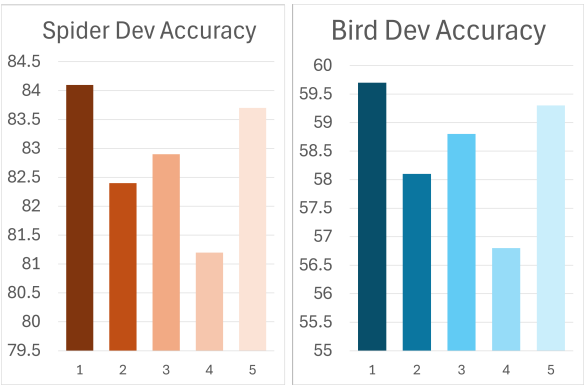

Figure 4: Spider Dev and Bird Dev Accuracy under weight configuration. 1–5 represent structural weight configurations: Heuristic (default), Reduced JOIN Weight, No Nested Penalty, Equal Weights, and High WINDOW Bias.

flecting the relative difficulty of SQL components when designing structure-aware curricula.The x-axis denotes different structure weight configurations used in the scoring function $S(x)$. "Heuristic" refers to manually assigned weights; "Reduced JOIN" lowers JOIN difficulty; "No Nested Penalty" ignores NESTED SELECT complexity; "Equal Weights" removes structural differentiation; "High WINDOW Bias" overemphasizes WINDOW operations.Interestingly, we observe that the Equal Weights variant—despite flattening all structural distinctions—still achieves performance comparable to the heuristic configuration, particularly on the Bird dataset. We hypothesize that this is due to the model's inherent ability to learn structural priors from large-scale pretraining, as well as Bird's higher baseline complexity, which may diminish the impact of structure-aware scheduling. Nonetheless, the heuristic setup consistently yields the best results, confirming the value of aligning training stages with compositional complexity.

Variant	JOIN	NESTED	WINDOW	Spider Dev	Bird Dev
Heuristic (default)	1.5	1.5	2.0	84.1	59.7
Reduced JOIN Weight	1.0	1.5	2.0	82.4	58.1
No Nested Penalty	1.5	0.5	2.0	82.9	58.8
Equal Weights	1.0	1.0	1.0	81.2	56.8
High WINDOW Bias	1.5	1.5	3.0	83.7	59.3

Table 5: Ablation on SQL structure in $S(x)$.

4.6 Structure-Specific Evaluation

To better understand the source of SAC-SQL's improvements, we conduct a structure-level breakdown of execution accuracy across different SQL component categories. Specifically, we classify evaluation samples according to whether they contain certain key structural elements such as `JOIN`, `HAVING`, and `NESTED SELECT`, and then compute execution accuracy within each subset. This analysis reveals how well models generalize to various levels of logical and compositional complexity.

Our results show that SAC-SQL exhibits substantial gains on structurally rich queries compared to the SFT baseline. For instance, on the Spider development set, SAC-SQL-13B improves execution accuracy on queries involving `JOIN` from 70.2% (SFT) to 78.4%, and on `GROUP BY` queries from 64.3% to 75.9%. The improvement is even more pronounced for queries containing `HAVING` clauses and `NESTED SELECT` subqueries, where the SFT model frequently fails to generate valid outputs. SAC-SQL achieves 68.7% execution accuracy on `HAVING` queries (vs. 52.1% for SFT), and 64.1% on `NESTED SELECT` queries (vs. 49.0%).

Similar trends are observed on the Bird dataset, where SAC-SQL demonstrates enhanced robustness to long, compositional question forms and schema-rich queries. In particular, the model exhibits fewer clause ordering errors and better handling of multi-step logical constraints, which are prevalent in Bird but underrepresented in Spider.

These improvements can be attributed to the structural curriculum design, which gradually exposes the model to increasingly complex SQL forms during training. This enables SAC-SQL to not only memorize isolated SQL components but also learn how to compose and integrate them in semantically correct ways. The result is a model that maintains high performance across both simple and challenging structural categories—something that standard SFT fails to consistently achieve.

Please refer to Appendix for ablation study (Appendix A) and deep discussions (Appendix B).

5 Conclusion

In this paper, we propose SAC-SQL, a model trained synthetic training samples followed by a structure-aware curriculum learning framework for enhancing SQL generation. It has integrated high-quality synthetic data, a unified difficulty scoring function, and phase-wise curriculum scheduling to guide the model through progressively more complex SQL structures. We have introduced SAC-SQL that achieves state-of-the-art execution accuracy on the Spider and Bird benchmarks, even surpassing several closed-source models such as GPT-4 in structurally demanding scenarios.

Limitations

While SAC-SQL achieves strong performance through structure-aware curriculum learning, several limitations remain. First, the curriculum schedule relies on a fixed difficulty scoring function that may not generalize well across domains or adapt dynamically during training. Although we incorporate model uncertainty into the scoring function, it remains statically defined prior to training and does not evolve with the model's competence. Second, the synthetic dataset, though structurally diverse, is generated using prompting heuristics and LLMs that may introduce distributional bias or lack realism compared to user-generated queries. Future work could explore human-in-the-loop or execution-guided data synthesis to enhance fidelity. Integrating curriculum learning with schema-aware pretraining or constrained decoding may further boost robustness. Finally, our framework assumes access to sufficient compute resources for phase-wise training, which may limit applicability in low-resource settings.

References

Rohan Anil, Andrew M. Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, Eric Chu, Jonathan H. Clark, Laurent El Shafey, Yanping Huang, Kathy Meier-Hellstern, Gaurav Mishra, Erica Moreira, Mark Omernick, Kevin Robinson, and 109 others. 2023. Palm 2 technical report. Preprint, arXiv:2305.10403.

Anthropic. 2023. Model card and evaluations for claude models.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, Binyuan Hui, Luo Ji, Mei Li, Junyang Lin, Runji Lin, Dayiheng Liu, Gao Liu, Chengqiang Lu, Keming Lu, and 29 others. 2023. Qwen technical report. arXiv preprint arXiv:2309.16609.

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. 2009. Curriculum learning. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, page 41–48, New York, NY, USA. Association for Computing Machinery.

Yinpei Dai, Hangyu Li, Yongbin Li, Jian Sun, Fei Huang, Luo Si, and Xiaodan Zhu. 2021. Preview, attend and review: Schema-aware curriculum learning for multi-domain dialog state tracking. Preprint, arXiv:2106.00291.

Longxu Dou, Yan Gao, Mingyang Pan, Dingzirui Wang, Wanxiang Che, Dechen Zhan, and Jian-Guang Lou. 2022. Unisar: A unified structure-aware autoregressive language model for text-to-sql. Preprint, arXiv:2203.07781.

Yujian Gan, Xinyun Chen, Jinxia Xie, Matthew Purver, John R. Woodward, John Drake, and Qiaofu Zhang. 2021. Natural sql: Making sql easier to infer from natural language specifications. Preprint, arXiv:2109.05153.

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

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. Deepseek-coder: When the large language model meets programming – the rise of code intelligence. Preprint, arXiv:2401.14196.

Zijin Hong, Zheng Yuan, Hao Chen, Qinggang Zhang, Feiran Huang, and Xiao Huang. 2024. Knowledge-to-SQL: Enhancing SQL Generation with Data Expert LLM. In Findings of the Association for Computational Linguistics: ACL 2024.

Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to sql: where are we today? Proc. VLDB Endow., 13(10):1737–1750.

Siqi Kou, Lanxiang Hu, Zhezhi He, Zhijie Deng, and Hao Zhang. 2024. Cllms: Consistency large language models. Preprint, arXiv:2403.00835.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. Preprint, arXiv:2302.05965.

Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023b. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing. Preprint, arXiv:2301.07507.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin C. C. Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023c. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. Preprint, arXiv:2305.03111.

Marwa Naïr, Kamel Yamani, Lynda Said Lhadj, and Riyadh Baghdadi. 2024. Curriculum learning for small code language models. Preprint, arXiv:2407.10194.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. Preprint, arXiv:2303.08774.

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. Preprint, arXiv:2304.11015.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Binhua Li, Fei Huang, Luo Si, Qingshan Jiang, and Yongbin Li. 2023. Schema dependency-enhanced curriculum pre-training for table semantic parsing. Knowledge-Based Systems, 262:110264.

Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022a. A survey on text-to-sql parsing: Concepts, methods, and future directions. Preprint, arXiv:2208.13629.

Bowen Qin, Lihan Wang, Binyuan Hui, Ruiying Geng, Zheng Cao, Min Yang, Jian Sun, and Yongbin Li. 2022b. Linking-enhanced pre-training for table semantic parsing. Preprint, arXiv:2111.09486.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. 2024. Direct preference optimization: Your language model is secretly a reward model. Preprint, arXiv:2305.18290.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2023. Exploring the limits of transfer learning with a unified text-to-text transformer. Preprint, arXiv:1910.10683.

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Romain Sauvestre, Tal Remez, Jérémy Rapin, Artyom Kozhevnikov, Ivan Evtimov, Joanna Bitton, Manish Bhatt, Cristian Canton Ferrer, Aaron Grattafiori, Wenhan Xiong, Alexandre Défossez, and 7 others. 2024. Code llama: Open foundation models for code. Preprint, arXiv:2308.12950.

Torsten Scholak, Nathan Schucher, and Dzmitry Bahdanau. 2021. PICARD: Parsing incrementally for constrained auto-regressive decoding from language models. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing, pages 9895–9901, Online and Punta Cana, Dominican Republic. Association for Computational Linguistics.

Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. 2024. A survey on employing large language models for text-to-sql tasks. Preprint, arXiv:2407.15186.

Ruoxi Sun, Sercan Ö. Arik, Alex Muzio, Lesly Miculicich, Satya Gundabathula, Pengcheng Yin, Hanjun Dai, Hootan Nakhost, Rajarishi Sinha, Zifeng Wang, and Tomas Pfister. 2024. Sql-palm: Improved large language model adaptation for text-to-sql (extended). Preprint, arXiv:2306.00739.

Xin Wang, Yudong Chen, and Wenwu Zhu. 2021. A survey on curriculum learning. Preprint, arXiv:2010.13166.

Jiaxi Yang, Binyuan Hui, Min Yang, Jian Yang, Junyang Lin, and Chang Zhou. 2024. 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, Bangkok, Thailand. Association for Computational Linguistics.

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. 2019. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. Preprint, arXiv:1809.08887.

Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. 2023. Act-sql: In-context learning for text-to-sql with automatically-generated chain-of-thought. Preprint, arXiv:2310.17342.

Qinggang Zhang, Junnan Dong, Hao Chen, Wentao Li, Feiran Huang, and Xiao Huang. 2024a. Structure guided large language model for sql generation.

Xuan Zhang, Gaurav Kumar, Huda Khayrallah, Kenton Murray, Jeremy Gwinnup, Marianna J Martindale, Paul McNamee, Kevin Duh, and Marine Carpuat. 2018. An empirical exploration of curriculum learning for neural machine translation. Preprint, arXiv:1811.00739.

Yiyun Zhang, Sheng'an Zhou, and Gengsheng Huang. 2024b. Se-hcl: Schema enhanced hybrid curriculum learning for multi-turn text-to-sql. IEEE Access, PP:1–1.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-SQL with distilled test suites. In Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 396–411, Online. Association for Computational Linguistics.

Alex Zhuang, Ge Zhang, Tianyu Zheng, Xinrun Du, Junjie Wang, Weiming Ren, Stephen W. Huang, Jie Fu, Xiang Yue, and Wenhu Chen. 2024. Structlm: Towards building generalist models for structured knowledge grounding. Preprint, arXiv:2402.16671.

A Ablation Study

To better understand each component in our proposed framework, we conduct a series of ablation experiments focusing on three key factors: the effect of curriculum learning, the influence of different components in the structure difficulty score $D(x)$, and the role of the scheduling strategy. The result of ablation studies in Table 6.

First, we evaluate the impact of curriculum learning by comparing SAC-SQL with a model trained using standard supervised fine-tuning (SFT) on the same synthetic dataset, but without any structure-aware phase scheduling. This ablated model shares the same architecture and training data as SAC-SQL but is trained on the full dataset from the beginning without curriculum partitioning. We observe that removing curriculum training leads to a consistent drop in execution accuracy across both datasets—particularly on structurally complex queries. On the Spider test set, accuracy drops from 84.1% to 78.7%, and on Bird, from 59.7% to 50.5%. This confirms that gradually increasing structural difficulty during training improves generalization, especially for challenging SQL patterns.

Second, we ablate components of the difficulty scoring function $D(x)$. In particular, we set the model-side uncertainty coefficient $\delta = 0$. This variant results in a performance degradation of 1.8% on Spider and 2.3% on Bird, with the greatest impact observed on structurally ambiguous or long-tail queries. These results underscore the importance of incorporating model confidence into the difficulty estimate, allowing the curriculum schedule to account not only for SQL structure but also for internal generation uncertainty.

Lastly, we examine the importance of structure-aware scheduling by replacing it with a randomized curriculum variant. In this setting, training samples are uniformly shuffled and partitioned into four curriculum phases of equal size, independent of $D(x)$. While this variant avoids catastrophic forgetting and achieves slightly better performance than baseline SFT, it still underperforms compared to our full structure-aware curriculum model. This demonstrates that the gains are not simply due to data reordering, but depend on the alignment between curriculum phase boundaries and the structural complexity of the SQL queries.

These ablations demonstrate that both the curriculum schedule and the structure-aware scoring function—particularly its inclusion of model-side uncertainty—are essential to the success of SAC-SQL. The performance gains observed are not incidental but arise from a principled integration of structural difficulty into the training dynamics.

Table 6: Ablation Study on Spider and Bird Datasets

Model Variant	Spider	Bird
SAC-SQL (Full Model)	84.1	59.7
w/o Curriculum Learning	78.7	50.5
w/o Model Uncertainty ($\delta = 0$)	82.3	57.4
Randomized Curriculum	80.5	54.8

B Discussion

The strong empirical results of SAC-SQL confirm the value of structure-aware curriculum learning for Text-to-SQL tasks, especially when applied to open-source language models. A key reason for its effectiveness lies in the alignment between the model's learning trajectory and the compositional structure of SQL itself.(Qin et al., 2022b) Unlike flat training paradigms that treat all samples as equal, our curriculum framework introduces a progressively challenging environment, enabling the model to first acquire syntactic fluency before confronting deeper structural dependencies. This learning dynamic mirrors human cognitive acquisition in complex domains and results in improved generalization on SQL forms.

Another important aspect is the interaction between structural curriculum learning and schema linking. While the latter focuses on grounding language to database content—mapping tokens to tables, columns, and values—curriculum learning operates at a higher level, shaping the model's ability to combine and organize those components into valid programs(Zhang et al., 2024b). In our setting, these two approaches are not mutually exclusive but complementary: SAC-SQL's curriculum prepares the model to better compose valid queries, while existing techniques can further enhance its schema awareness. We believe that integrating schema-linking modules within our curriculum framework could yield even stronger results, especially in low-resource scenarios.

Finally, the principles behind SAC-SQL extend naturally to other structured generation tasks beyond SQL. Tasks such as natural language to code (NL2Code), knowledge base querying (NL2Query), and even Tabular Question Answering share the same core challenge: map-

ping unstructured input into logical, executable forms.(Naïr et al., 2024) In all of these settings, output structures vary in complexity, and learners benefit from phased exposure to that complexity. Our framework provides a general recipe for such settings: define a domain-specific structure difficulty function, partition data accordingly, and train models with difficulty-aligned schedules. We expect curriculum-guided LLM training to play a growing role in structured reasoning tasks across modalities and domains.