
KNOWLEDGE DISTILLATION WITH STRUCTURED CHAIN-OF-THOUGHT FOR TEXT-TO-SQL

Khusbboo Thaker¹, Yony Bresler¹

¹Crater Labs, Toronto, Canada

December 22, 2025

ABSTRACT

Deploying accurate *Text-to-SQL* systems at the enterprise level faces a difficult trilemma involving cost, security and performance. Current solutions force enterprises to choose between expensive, proprietary Large Language Models (LLMs) and low-performing Small Language Models (SLMs). Efforts to improve SLMs often rely on distilling reasoning from large LLMs using unstructured Chain-of-Thought (CoT) traces, a process that remains inherently ambiguous. Instead, we hypothesize that a formal, structured reasoning representation provides a clearer, more reliable teaching signal, as the *Text-to-SQL* task requires explicit and precise logical steps. To evaluate this hypothesis, we propose *Struct-SQL*, a novel Knowledge Distillation (*KD*) framework that trains an SLM to emulate a powerful large LLM. Consequently, we adopt a query execution plan as a formal blueprint to derive this structured reasoning. Our SLM, distilled with structured CoT, achieves an absolute improvement of 8.1% over an unstructured CoT distillation baseline. A detailed error analysis reveals that a key factor in this gain is a marked reduction in syntactic errors. This demonstrates that teaching a model to reason using a structured logical blueprint is beneficial for reliable SQL generation in SLMs.

1 Introduction

Text-to-SQL (NL2SQL or text2sql) has the potential to democratize data access [1]. The field has seen substantial performance advancements driven by the advent of Large Language Models (LLMs) [2]. These models enhance the capabilities of natural language interfaces for databases by automatically translating natural-language user questions into SQL queries [3]. Nevertheless, widespread adoption in enterprises remains challenging due to a difficult trade-off among three interdependent factors: cost, security, and performance. This challenge can be understood as an *Adoption Trilemma*.

- **Cost:** High-performing models typically require significant computational resources, leading to high operational costs, whether using proprietary APIs or privately hosted LLMs [2].
- **Security:** Relying on external APIs raises prominent security concerns, as transmitting potentially sensitive database schemas and sample records to third-party providers is often unacceptable in enterprise settings [4].
- **Performance:** Selecting open-source models for local, private deployment to address cost and security issues often leads to the use of Small Language Models (SLMs), which typically lack adequate zero-shot accuracy for complex real-world queries [5].

To identify the limitations of current approaches, we examine the impact of recent advances in reasoning-driven prompting on *Text-to-SQL* performance. A notable portion of the recent performance gain in *Text-to-SQL* using LLMs can be attributed to in-context learning (ICL) [6]. ICL-based methods, particularly those that use decomposition and multi-step reasoning, have demonstrated substantial gains in performance on the *Text-to-SQL* task [3]. A prominent ICL technique for this is Chain-of-Thought (CoT), which encourages models to think step-by-step [7, 8]. Such decomposition strategies are beneficial for generating complex SQL queries. For instance, *DAIL-SQL* [9] enhances CoT through context-aware examples, while *DIN-SQL* [10] and *Divide-and-Conquer CoT* (DC-CoT) [11] improve accuracy by breaking questions into intermediate sub-queries. Building on this logical foundation, the Query Plan CoT

(*QP-CoT*) guides the model through steps that mirror a database execution plan, simulating the logical process by which databases execute queries. By following this structured path, *QP-CoT* ensures that the SQL generation path aligns with the inherent logic of query execution [11]. Although these reasoning techniques achieve considerable success, their effectiveness is observed almost exclusively with large LLMs. This reliance on large LLMs constitutes a significant limitation: these methods depend on the very models that exacerbate the cost and security challenges, thereby limiting their applicability in resource-constrained enterprise settings and contributing to the ‘Adoption Trilemma’.

The performance limitations of SLMs, suitable for private deployment, are particularly pronounced in the *Text-to-SQL* domain. Standard benchmarks highlight a significant gap between large models and smaller open-source alternatives. For example, on the BIRD mini-dev benchmark, LLMs such as *GPT-4o* and *Claude* ($\geq 150\text{B}$ parameters) achieve execution accuracies between 30% and 45%, whereas widely used SLMs such as *Mistral*, *Mixtral* and *Qwen2.5 Coder* ($\sim 7\text{B}$ parameters) achieve only 4% to 12%¹. This paper reveals that this severe performance degradation in SLMs persists even when advanced reasoning prompts are employed, despite their effectiveness in larger LLMs. This finding is consistent with recent literature, suggesting that large LLMs can adhere to the schema as long as the schema fits within their context window, a capability not observed in SLMs [12]. In our experiments, the SLM (*Qwen3-4B-Instruct-2507*) prompted with *QP-CoT* fails primarily due to inadequate schema adherence, with a pronounced tendency toward schema hallucination, often generating non-existent tables or columns (see Figure 3b). Thus, while large LLMs benefit from structured reasoning strategies, SLMs do not internalize these logical decompositions. This breakdown in structured reasoning motivates an investigation into whether the *structure of reasoning* itself can be effectively transferred from large LLMs to SLMs.

To bridge this performance gap, Knowledge Distillation (*KD*) [13], which transfers reasoning ability from a capable teacher model, is a key strategy [2]. *KD* is a model compression technique in which a smaller “student” model is trained to mimic the behavior of a larger, pretrained “teacher” model. Beyond compression, *KD* enables the transfer of complex reasoning [14] and logical skills [15, 16]. Through *KD*, our aim is to build SLMs for *Text-to-SQL* that deliver accuracy comparable to larger LLMs while meeting enterprise cost and security requirements through private deployment. This work argues that *KD* is a promising approach in addressing this trilemma, but its effectiveness depends on the type of reasoning being distilled.

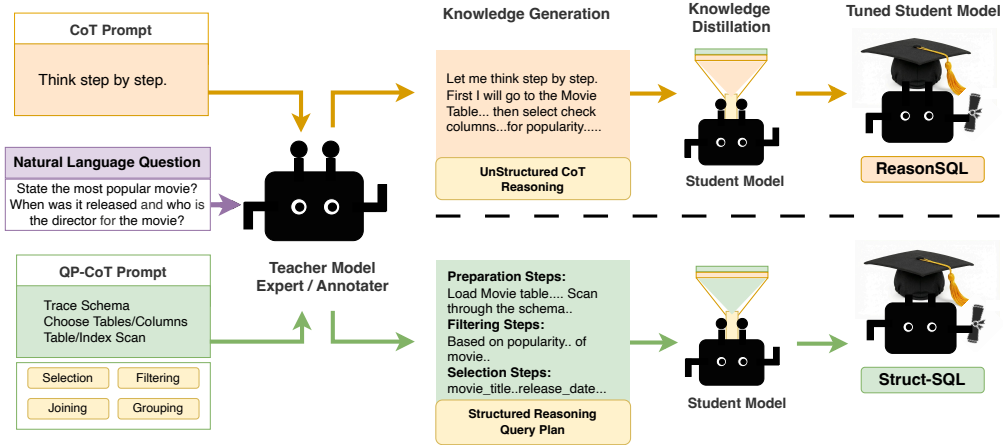


Figure 1: Unstructured vs. Structured Reasoning Distillation. The figure contrasts the two methods of reasoning distillation: (Top) Unstructured Distillation (*ReasonSQL*), which relies on a free-form CoT prompt, with (Bottom) the proposed Structured Distillation (*Struct-SQL*), which uses a *QP-CoT* prompt to generate a structured logical blueprint. The Teacher Model’s output serves as the supervisory signal for tuning the Student Models.

Given that the core bottleneck for SLMs is replicating the teacher’s complex reasoning, the central question for *KD* is identifying the optimal form of the reasoning signal to distill. Building on the success of CoT prompting [7], one approach is to distill the teacher’s natural language reasoning traces [14]. This approach is exemplified by *ReasonSQL*, where the student model is trained on the teacher’s intermediate CoT steps in addition to the final SQL query. This unstructured approach provides a better knowledge signal than finetuning on SQL queries alone and has been shown to

¹Performance on BIRD mini-dev, see https://github.com/bird-bench/mini_dev

improve model accuracy [17]. However, we argue that the structure of the reasoning signal itself is critical for effective distillation. We hypothesize that distilling knowledge using a more formal, structured representation of the reasoning process that directly reflects the logical steps of query execution could provide a clearer, less ambiguous, and more beneficial supervisory signal for distilling SLMs than unstructured CoT explanations.

To evaluate this hypothesis, we introduce *Struct-SQL*, a framework for distilling structured reasoning. Figure 1 shows the *KD* workflow, contrasting the proposed structured reasoning approach against the standard unstructured CoT method. Within this framework, a state-of-the-art (SOTA) Teacher Model is used to generate a *QP-CoT* trace, which formally decomposes the query into a logical execution plan [11]. This structured plan, together with the generated SQL query, constitutes the supervisory signal. A student model is then trained to replicate the entire structured output sequence (query plan, SQL). The formal query plan serves as a clear, hierarchical blueprint that guides the student model in learning the precise, logical steps of query construction, from schema linking and join-path selection to aggregation and filtering. During inference, this structured reasoning is retained: the student model is given the *QP-CoT* prompt to autonomously generate the query plan before synthesizing the final SQL query.

We perform an extensive comparative analysis on the mini-dev BIRD benchmark [5], evaluating our *Struct-SQL* framework against key baselines. Our contributions are as follows.

- We are the first to systematically evaluate the impact of *KD* using a structured reasoning signal for *Text-to-SQL*.
- We provide a comprehensive error analysis, showing that the improvement in *Struct-SQL* results from a marked reduction in syntactic errors (e.g., schema hallucination), demonstrating that the structured signal provides a clearer curriculum.
- We validate the generalization of our framework through an ablation study on two different SLMs.
- We release the *KD* dataset of 1,300 structured reasoning traces, the *Struct-SQL* model, prompts and the code ².

Following this introduction, Section 2 details the methodology. Section 3 presents the results, which are contextualized by the related work in Section 4, and the paper concludes in Section 5.

2 Methodology

2.1 Problem Formulation

The standard *Text-to-SQL* task involves mapping a natural language question Q and a database schema S to an executable SQL query Y_{Gold} . Formally, given an input pair (Q, S) , the goal is to learn a model M such that $M(Q, S) = \hat{Y}$, where \hat{Y} represents the predicted query intended to give the same result as the ground truth Y_{Gold} . Our work focuses on transferring the capabilities of a large, high-performance Teacher Model, M_T , to a smaller, more efficient Student Model, M_S , through *KD*. We formulate the distillation task as learning the Teacher’s intermediate reasoning steps, R_T , in addition to its final output, Y_T . Let $Z_T = R_T \oplus Y_T$ represent the complete output sequence. Training is conducted over a distillation dataset $\mathcal{D}_{\text{DISTILL}}$. We employ a standard sequence completion loss, minimizing the negative log-likelihood of the teacher-generated text Z_T by updating the parameters θ of the student model:

$$\mathcal{L}_{KD} = - \sum_{(Q, S, Z_T) \in \mathcal{D}_{\text{DISTILL}}} \log P_{M_S(\theta)}(Z_T | Q, S) \quad (1)$$

We instantiate R_T either as an unstructured CoT or a structured *QP-CoT* to test the hypothesis that a structured reasoning trace provides a more effective supervisory signal.

2.2 Structured CoT via Query Execution Plan

Our structured CoT strategy is inspired by a database engine’s query execution plan. A query execution plan defines the precise sequence of steps that a database follows to access and manipulate data, often generated using an *EXPLAIN* command. We adopt this ICL prompt strategy from a recent work on a Query Execution Plan-based prompting strategy, *QP-CoT* [11]. Instead of a free-form explanation, the Teacher Model is prompted to generate a query plan along with the final SQL query. As shown in Figure 2, this plan decomposes the query into a sequential execution flow that explicitly performs selections, filters, joins, and groupings through step-by-step table scanning and data manipulation, providing a systematic signal for the student model. Although the query plan has been explored as a prompting technique [11], the key contribution here is to demonstrate the effectiveness of this structured reasoning as the primary teaching signal within a *KD* framework.

²<https://github.com/craterlabs/Struct-SQL-Distillation/>

```

** Preparation Steps:**
1. Initialize the process: Start preparing to execute the query.
2. Prepare storage: Set up storage space (registers) to hold temporary results, initializing them to NULL.
3. Open the movies table: Open the movies table so we can read from it.
** Finding the Most Popular Movie:**
1. Start reading the movies table: Move to the first row in the movies table.
2. Check the movie popularity: Look at the `movie_popularity` column of the current row in `movies`. If it's not the highest value, skip this row.
3. Identify the matching row: Store the `movie_id` of this movie entry. This `movie_id` is for the most popular movie.
** Retrieving Movie Details:**
1. Prepare to retrieve details: If a `movie_id` is found from the previous step, prepare to retrieve details.
2. Start reading the movies table again: Move to the row in `movies` where `movie_id` matches the identified `movie_id`.
3. Select the required columns: Retrieve the `movie_title`, `movie_release_year`, and `director_name` for the identified movie.
** Delivering the Result:**
1. Output the result: Output the movie title, release year, and director name.
2. End the process: Stop the query execution process

```

Figure 2: Sample structured query plan for the input "State the most popular movie? When was it released and who is the director for the movie?"

2.3 Experimental Setup

The experiments use the SQLite-based *BIRD benchmark* for the data [5]. The BIRD training dataset is used for model training, while the BIRD mini-dev dataset serves for evaluation. The Teacher Model (M_T), GPT-4o (OpenAI), is selected due to its leading performance on the BIRD mini-dev dataset and functions as an oracle to generate high-quality query plans. The Student Model (M_S) is *Qwen3-4B-Instruct-2507* (Alibaba Cloud). It was chosen for its strong performance-to-size ratio, which enables low-latency, private deployment and thus addresses the Adoption Trilemma [18]. All models utilize a single-pass inference mechanism and operate without multi-agent collaboration, self-consistency checks or external correction loops.

2.3.1 Model Configurations

Pretrained Models: These configurations evaluate the model’s intrinsic few-shot ICL capability using *QP-CoT* without parameter updates.

- *Teacher Model:* GPT-4o (M_T) to establish the upper bound.
- *Student Model:* Qwen3-4B-Instruct-2507 to establish the lower bound.

Tuned Student Models: These configurations evaluate the Student Model after tuning on specific signals.

- *FN-Gold* The Student Model was finetuned using the Gold SQL query (Y_{Gold}) on the BIRD training dataset, with a basic system instruction that directly maps natural language to SQL [10].
- *ReasonSQL* (unstructured *KD*): The Student Model was finetuned on the complete Teacher sequence $Z_T = R_{\text{CoT}} \oplus Y_T$, where R_{CoT} is the free-form CoT rationale [17]. This configuration serves as the established baseline for evaluating the efficacy of standard unstructured reasoning distillation.
- *Struct-SQL* (Structured *KD*): The Student Model was finetuned on the entire Teacher sequence $Z_T = R_{\text{QP-CoT}} \oplus Y_T$, where $R_{\text{QP-CoT}}$ is the formal query plan CoT trace. This trace represents the structured logical blueprint designed for the core hypothesis.

During inference, all models use the *QP-CoT* prompt structure except for *ReasonSQL*, which employs the unstructured *CoT* prompt.

2.3.2 Distillation Dataset Construction

The *KD* datasets were constructed using an active generation and filtering methodology to maximize data quality and the diversity of query complexity. To initiate the process, the databases in the corpus are partitioned into a 75% "in-domain" (ID) database pool and a 25% "out-of-domain" (OOD) database pool, based on unique database identifiers. This partitioning ensures that the ID pool serves as the source for all training data and the in-domain validation set, while the OOD pool is used exclusively for the out-of-domain validation set, guaranteeing a robust measure of the trained model’s generalization capabilities.

We then applied stratified, success-based sampling. The ID corpus is iterated using predefined SQL structure and syntax categories to achieve target distributions for the training set (1,000 samples) and the in-domain validation set (150 samples). These categories, as shown in Table 1, are differentiated by the complexity of the SQL structure, including single-table queries, queries with subqueries, queries with joins/set operations and queries that combine both joins/set operations and subqueries. For each candidate, inference was performed using *GPT-4o*. A sample is admitted only if the generated SQL is both syntactically valid and yields the correct result upon execution. New samples are generated until 1,000 successful training samples and 150 ID validation samples are obtained. The OOD validation set (150 samples) is constructed using the same sampling and validation process, but applied exclusively to the segregated OOD database pool to ensure that no schema overlaps. For both *Struct-SQL* and *ReasonSQL*, we used the same data generation pipeline to ensure a controlled, methodologically fair comparison. The datasets differ only in the format of the supervisory signal: unstructured CoT traces for *ReasonSQL* versus structured *QP-CoT* for *Struct-SQL*.

Complexity Category	Training Count	ID Val Count	OOD Val Count
Single Table Queries	295 (29.50%)	37 (24.67%)	46 (30.67%)
Subquery (no join or set operations)	229 (22.90%)	39 (26.00%)	31 (20.67%)
With JOINS / Set Ops (no Subquery)	398 (39.80%)	57 (38.00%)	60 (40.00%)
JOINS / Set Ops and Subquery	78 (7.80%)	17 (11.33%)	13 (8.67%)
Total Successful Samples	1000	150	150

Table 1: Distribution of Query Complexity Across Distillation Datasets. The data is partitioned by unique database identifiers to evaluate generalization. ID Val = In-Domain Validation; OOD Val = Out-of-Domain Validation (databases unseen in training).

2.4 Implementation and Evaluation Details

2.4.1 Post-Training Details

FN-Gold, *ReasonSQL*, and *Struct-SQL* were finetuned using Parameter-Efficient Fine-Tuning (PEFT) using Quantized Low-Rank Adaptation (QLoRA) [19] to improve training efficiency. *PEFT* was chosen for its proven effectiveness in adapting SLMs to new domains with small training datasets. Furthermore, *PEFT* has been shown to yield more stable models and mitigate the risk of catastrophic forgetting of foundational knowledge. *QLoRA* allows recovery of the full 16-bit finetuning task performance even when the base model is loaded at 4-bit precision [20]. The *KD* approach keeps the original model parameters θ_{BASE} fixed and only updates a small set of *LoRA* adapter parameters θ_{ADAPTER} . We followed the QLoRA methodology [19] and applied adapters to all linear layers of the Transformer architecture. Based on preliminary testing on a subset of the development data, we selected $r = 64$ and an alpha scaling of $\alpha = 128$ to ensure robust adaptation. Optimization was performed using AdamW with a learning rate of 10^{-4} , a maximum input length of 15,000 tokens, and a generation limit of 1,500 tokens. We utilized a batch size of 6 for *ReasonSQL* and *Struct-SQL*, compared to 15 for the *FN-Gold* baseline on an NVIDIA H200 GPU. All models minimized completion loss. To balance in-domain accuracy with out-of-domain generalization, we used an early-stopping strategy (patience=8) that monitored the aggregated validation loss across both the ID and OOD validation sets.

2.4.2 Evaluation Metrics

The primary metric is Execution Accuracy (EX), which measures the percentage of generated SQL queries that execute without error and return the same result set as the ground-truth SQL query on the BIRD mini-dev set. A detailed analysis of model failure modes reveals why and where certain models perform best. To facilitate analysis, we categorize failures into three distinct types as defined in Table 2. This classification establishes a severity hierarchy for *Text-to-SQL* tasks: starting with the most severe Generation Failure (GEN), where the model produces no recognizable SQL output; followed by Syntactic Failure (SYN), indicating an unexecutable query due to grammar or schema errors; and finally,

Logical Failure (SEM), representing a query that is syntactically correct but semantically inaccurate. Recognizing this progression is beneficial for discerning the specific challenges and improvements associated with each model.

Error Type	Subcategory	Description
Semantic Errors (SEM)	Incorrect Columns/Rows	Wrong columns/row count
	Value Mismatch	Incorrect data values
	Empty Output	Empty result set
Syntactic Errors (SYN)	No Such Column/Table	Non-existent table/column
	Keyword Issue	Incorrect or misplaced SQL keyword
	Syntax/Clause Order	Incorrect clause order, missing parentheses, etc
Generation Error (GEN)	N/A	No SQL output

Table 2: The proposed hierarchical error taxonomy. This table categorizes *Text-to-SQL* errors in order of increased severity and is used for subsequent failure analysis.

3 Results

3.1 Overall Performance

The execution accuracy of *Struct-SQL*, compared to all baselines on the BIRD mini-dev dataset, provides strong support for the central hypothesis. As summarized in Table 3, distilling structured reasoning results in significantly better performance compared to unstructured distillation, *ReasonSQL* and traditional finetuning *FN-Gold*. The native Student Model’s performance (17.0%) illustrates the Performance Trade-Off in the Adoption Trilemma, indicating that an SLM is insufficient for production deployment without intervention. *Struct-SQL* achieved an 8.1-point absolute improvement from 36.90% to 45.00% over the *ReasonSQL* baseline. This improvement enables the Student Model to cover 84% of the Teacher Model’s execution accuracy.

Model	Prompt	Overall		EX by Difficulty		
		EX	Avg. Tokens	Simple	Moderate	Challenging
Teacher (<i>GPT-4o</i>)	<i>QP-CoT</i>	53.60%	298 ± 96	68.24%	52.00%	36.27%
Student Model	<i>QP-CoT</i>	17.00%	1465 ± 136	34.45%	9.60%	9.80%
Tuned Student Models						
<i>FN-Gold</i>	<i>QP-CoT</i>	34.30%	198 ± 257	45.94%	32.80%	20.58%
<i>ReasonSQL</i>	CoT	36.90%	99 ± 99	49.32%	33.20%	27.45%
<i>Struct-SQL</i> (Ours)	<i>QP-CoT</i>	45.00%	362 ± 201	65.54%	40.4%	25.49%

Table 3: Overall and difficulty-wise execution accuracy (EX) on the BIRD mini-Dev set, along with average inference tokens (Avg. Tokens). *Struct-SQL* achieves the highest EX among all tuned student models, particularly for simpler and moderate queries. An 8.1-point EX improvement over *ReasonSQL* requires 3.6 times more tokens.

3.2 Analysis of Model Failures

Figure 3 presents hierarchical pie charts. The inner ring displays top-level results that depict success versus categorized error types, while the outer rings further specify subcategories of failures, as defined in Table 2. For successful queries, the outer ring categorizes results by the difficulty level of the SQL queries in the BIRD dataset. As visually summarized in the hierarchical pie charts, a key dichotomy emerges between the Teacher and other baselines, primarily distinguished by their dominant error types. The Teacher Model (M_T), shown in Figure 3a, achieved the highest success rate. Its failures were predominantly semantic errors (~30%), indicating challenges with logical complexity. The Student Model (M_S), shown in Figure 3b, had a higher number of syntactic errors, including schema hallucination, underscoring a fundamental inability to adhere to the database schema before post-training.

The standard finetuning baseline (*FN-Gold*) reflects the performance achieved by training solely on the gold SQL (Y_{Gold}), without incorporating intermediate reasoning. As shown in Figure 3c, this approach focused its gains almost entirely on reducing semantic errors (from 53.8% to 33.0%, a 20.8-point drop) of the Student Model M_S , but failed to address the

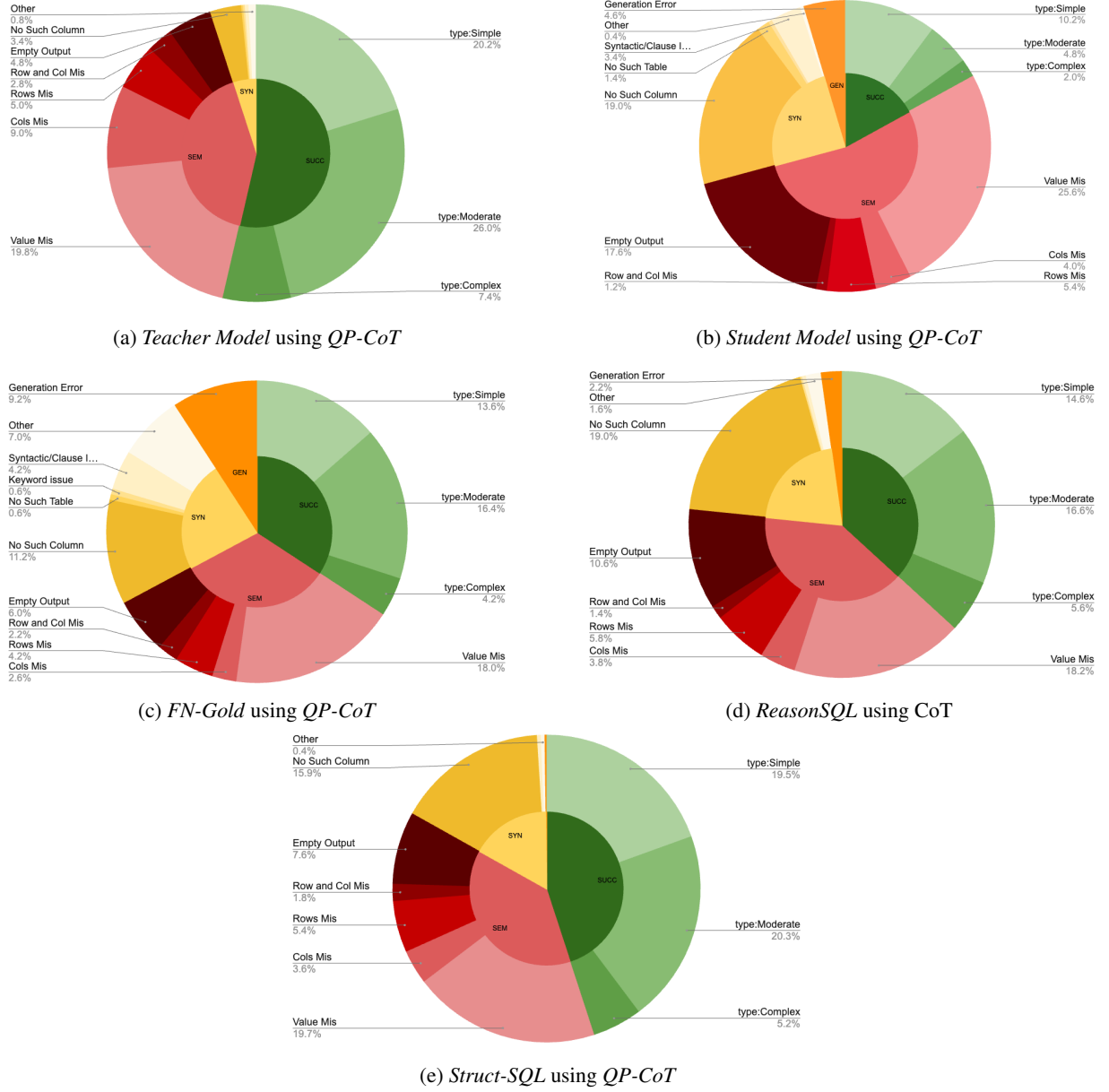


Figure 3: Compared to the Teacher Model (a), both the Student Model (b) and the *FN-Gold* (c) exhibit substantially lower performance, primarily due to high syntactic errors. The unstructured distillation baseline *ReasonSQL* (d) improves upon both the Student Model and *FN-Gold*. *Struct-SQL* (e) achieves the highest success rate among all tuned student models.

syntactic bottleneck: overall syntactic errors remained nearly unchanged (24.6% to 23.6%). These results confirm that training on only the final query output is insufficient for learning strict SQL grammar.

The unstructured distillation (*ReasonSQL*) baseline evaluates distillation through an unstructured reasoning trace (R_{CoT}). As shown in Figure 3d, this approach achieved an execution accuracy of 36.9%, higher than that of M_S and *FN-Gold*, indicating that unstructured *CoT* reasoning provides a better distillation signal than both the naive model and *FN-Gold*. *ReasonSQL* significantly reduced the rate of generation errors compared to the naive Student Model M_S , suggesting that unstructured distillation enhanced the stability of the model. Furthermore, it reduced semantic errors from 53.8% to 39.8% (a 14.0-point reduction) and reduced syntactic errors from 24.6% to 21.2% (a 3.4-point reduction), primarily by improving basic schema linking and eliminating ‘No Such Table’ errors. These findings align

with previous research demonstrating that CoT-based distillation improves accuracy [15] and further clarify its specific advantages.

As established above, unstructured distillation *ReasonSQL* outperformed both the naive Student Model and standard finetuning *FN-Gold*. *Struct-SQL*, as shown in Figure 3e, outperformed *ReasonSQL*. By distilling the reasoning signal into a structured blueprint, *Struct-SQL* increased execution accuracy from 36.9% to 45.0% (an 8.1-point improvement) compared to *ReasonSQL*. This improvement appeared across the error severity hierarchy. *Struct-SQL* nearly eliminated generation errors, reducing the rate from 2.2% to 0.4% (a 1.8-point reduction). For syntactic errors, although *ReasonSQL* outperformed the naive model, it did not enforce strict schema alignment. In contrast, *Struct-SQL* reduced the total syntactic error count from 21.2% in *ReasonSQL* to 16.8% (a 4.4-point reduction). This reduction included fewer ‘No Such Column’ hallucinations, 19.0% to 15.8% (a 3.2-point reduction) and the elimination of ‘Keyword Issues’, showing a more precise understanding of SQL syntax. For semantic errors, the structured model showed greater logical reliability, reducing ‘Empty Output’ errors from 10.6% in *ReasonSQL* to 7.6% (a 3.0-point reduction). Although this increased rigidity led to a minor trade-off, *ReasonSQL* retained a slight advantage in ‘Value Mismatches’ (18.2% vs. 19.6%), suggesting that the structured plan provides a more robust signal.

3.3 Fine-Grained Performance Analysis and Ablation Study

To investigate the precise source of performance of *Struct-SQL*, we move beyond aggregate metrics to examine the detailed differences between models.

Performance by SQL Construct: Assessing model performance across the spectrum of query formulations can be insightful; therefore, we analyzed execution accuracy for key SQL constructs, as shown in Figure 4a. The comparative analysis indicates that *Struct-SQL*’s performance gains are most pronounced in tasks that require aggregation and explicit structural decomposition. This is evidenced by its strong results on specific SQL operations. For queries requiring a GROUP BY clause, the *Struct-SQL* model (42.0% EX) outperformed both *FN-Gold* (33.8% EX) and *ReasonSQL* (34.3% EX), demonstrating that explicitly distilling the query execution plan effectively trains the model on the necessary aggregation formulation. An advantage was also observed for queries requiring an ORDER BY, where *Struct-SQL* showed a 2.8-point absolute improvement over *ReasonSQL*. However, the comparative analysis revealed a notable exception: for queries that require JOIN operations, *ReasonSQL* achieved the highest execution accuracy. In contrast, *Struct-SQL*’s score (25.5% EX) was lower, suggesting that the unstructured CoT trace provided a unique advantage for learning multi-entity linking. This area remains a core challenge for all models, as confirmed by the Teacher Model’s performance (36.4% EX). Furthermore, all baselines, including *Struct-SQL*, shared a weakness in handling queries that require Set Operations.

Model	Overall EX (%)	EX by Difficulty		
		Simple	Moderate	Challenging
Student Model	7.22	11.49	6.00	3.92
<i>ReasonSQL</i> (CoT)	25.10	40.54	20.08	12.75
<i>Struct-SQL</i> (Ours)	29.31	49.32	23.20	14.71

Table 4: Ablation study on *Mistral-7B-Instruct-v3.0* demonstrates that *Struct-SQL* continues to outperform unstructured *ReasonSQL* across different base models.

Evaluation of Knowledge Transfer: Gains vs. Losses: A Gains vs. Losses analysis was conducted to assess the quality of the distilled knowledge (Figure 4b). This approach compares queries where the Teacher was correct, but the Student Model failed (Losses, red segments) with cases where the Student Model was correct, but the Teacher Model was incorrect (Gains, green segments), representing true generalization. The results indicate that *Struct-SQL* demonstrates the highest net performance relative to the Teacher Model by exhibiting the fewest losses and the most gains among all student models. This confirms that the structured reasoning signal leads to better fidelity in knowledge transfer.

Performance State Transitions: Figure 4c illustrates the progression of queries from the baseline Student Model to the distilled *Struct-SQL*, highlighting the systematic conversion of error states into successful or unsuccessful output. *Struct-SQL* retained more than 81% of the initial successes and enhanced stability by reducing generation errors of the Student Model from 4.0% to 0.4%. The structured framework also corrected 41.3% of semantic errors (SEM → SUCC), confirming that the formal query plan addresses logical flaws. Additionally, over two-thirds of syntactic failures were upgraded: 29% were corrected (SYN → SUCC) and 41% shifted to semantic errors (SYN → SEM).

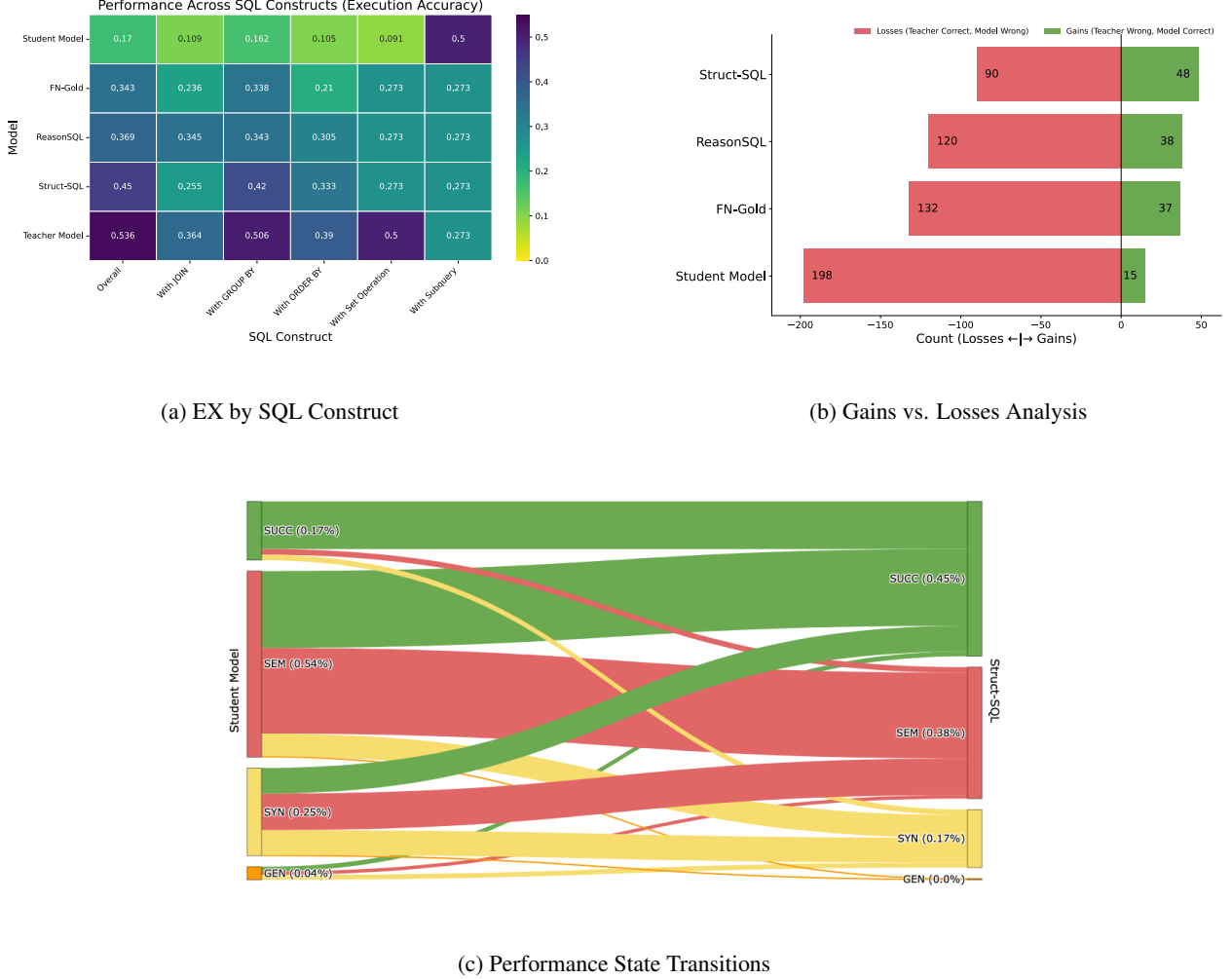


Figure 4: Detailed performance analysis. (a) Execution Accuracy across different SQL constructs, highlighting *Struct-SQL*’s proficiency in handling complex aggregations. (b) Gains vs. Losses analysis for baseline models relative to the Teacher model. The losses and gains are correlated with overall performance. (c) Performance State Transitions illustrates *Struct-SQL*’s effectiveness in converting severe errors (SYN, GEN) from the Student Model into direct successes or less severe errors (SEM).

Generalization on *Mistral-7B*: To validate the transferability of our framework to other architectures, we replicated our experiments on *Mistral-7B-Instruct-v3.0*. This model was specifically selected for its low zero-shot accuracy of 4.4% EX on BIRD mini-dev. As shown in Table 4, *Struct-SQL* shows generalizability, achieving a performance advantage (29.3% EX) over the Student Model (7.2% EX) and the *ReasonSQL* baseline (25.1% EX), confirming that the structured query plan offers a robust supervision signal independent of the base model.

Computational Efficiency: On a single H200 GPU, using *Qwen3-4B-Instruct-2507* as the Student Model, *Struct-SQL* converged using an early stopping strategy with a patience of 8 steps and a threshold of 0.001 in 2.24 epochs, requiring only 29.15 minutes. This training efficiency is comparable to that of the unstructured *ReasonSQL*, which required 25.24 minutes across 6.4 epochs with the same batch size, sample count and early stopping strategy. This 1,000-sample distillation strategy is also computationally tractable compared to traditional finetuning on larger datasets. For context, the *FN-Gold* baseline, trained on the entire 9,000+ sample BIRD dataset, required 110.57 minutes (4.33 epochs) to converge. This efficiency underscores the practical deployability of our structured distillation method in resource-constrained environments.

Official Test Evaluation: Verified by the BIRD benchmark team, our model achieves 60.42% EX³, ranking first among $\leq 4B$ models (as of December 22, 2025). This result is achieved using strict single-model inference with greedy decoding and no self-consistency.

4 Related Work

4.1 Text-to-SQL with In-Context Learning (ICL)

LLMs fundamentally redefined the field of *Text-to-SQL* [21], achieving substantial performance on challenging cross-domain benchmarks such as Spider [22] and BIRD [5]. ICL [6] has emerged as a dominant research paradigm, enabling LLMs to leverage their pre-training by processing instructions and examples for SQL generation [2]. Initially, ICL approaches utilized foundational zero-shot and few-shot paradigms [23, 24, 25, 26]. However, generating complex SQL queries that involve nested logic and multitable joins was found to require explicit external guidance [10]. As a result, the field pivoted toward decomposition, a multi-step reasoning tactic designed to improve accuracy and reliability [10, 27, 9, 28, 29]. As a form of ICL-based decomposition, basic CoT helped LLMs break down the task, follow a logical workflow and improve execution accuracy [29]. This progression led to sophisticated variants such as *DIN-SQL* [10], which decomposes the task into sequential stages (e.g., schema linking, query classification). Additional multi-stage architectural interventions have also been explored [27, 28]. Crucially, these approaches often require multiple LLM calls to intervene, correct, or gather intermediate data, leading to high latency and cost. To mitigate this, some approaches focused on optimizing efficiency by creating single-pass Structural CoT approaches [11, 29]. These methods instruct the LLM to follow a formalized logical blueprint to gain the necessary understanding of the schema and natural language before generating the SQL. Specialized methods, including *QP-CoT* and *QDecompose*, excel here, as they have been shown to produce better results than non-structured CoT [11, 9]. For example, *QP-CoT* provides few-shot query plan examples that require the model to first generate a detailed query plan before generating SQL code [11]. Although single-pass ICL performs well with large LLMs, its gains are limited when applied to SLMs [12] or require expensive finetuning [30, 8, 31, 32]. This work addresses this identified gap by introducing *Struct-SQL*, a novel *KD* method that enables single-pass ICL for SLMs.

4.2 KD for Reasoning Transfer

KD has evolved from a model compression technique to a method for complex knowledge transfer, called Skill Distillation, often used to transfer reasoning knowledge from one model to others or to facilitate self-improvement [13, 2]. Due to the significant resource demands of LLMs, recent research has focused on efficiently transferring its reasoning capabilities from a teacher LLM to a student SLM. This transfer is frequently grounded in *ICL* principles, demonstrating that learning from a few-shot demonstrations can be successfully distilled into the SLMs’ parameters [33]. The primary established approach to enable reasoning in SLMs is the Finetune-CoT paradigm [14], which uses unstructured CoT explanations of the teacher as a supervision signal. This distillation approach has been demonstrated to be effective with multi-step mathematical reasoning [16]. However, relying on unstructured CoT often leads SLMs to learn spurious correlations rather than deep causal features, thereby reducing robustness on complex data [34, 35]. Consequently, recent research increasingly favors methods that enforce structural decomposition and explicit causality. These approaches include creating constrained distillation pipelines, with the aim of removing ambiguous input context through advanced structural formats. For example, *SocraticCoT* provides an explicit structural format by guiding the model through reasoning using defined subquestion-solution pairs [35]. Alternatively, *Mixed Distillation* integrates the CoT with a formal verifiable Program-of-Thought reasoning to provide a less ambiguous supervision signal [34]. In the *Text-to-SQL* domain, structured *KD* has seen limited uptake. Previous studies, while validating the effectiveness of distillation in this area, have generally relied on less structured signals, such as schema-based finetuning to improve schema linking [3], the inclusion of enterprise-specific custom examples [4], and unstructured CoT reasoning transfer [17]. The proposed *Struct-SQL* framework systematically evaluates whether structure-based reasoning transfer, specifically *QP-CoT*, improves the distillation for *Text-to-SQL*.

5 Conclusion and Future Research

This work presents *Struct-SQL*, a *KD* framework that transfers structured reasoning (*QP-CoT*) from a Teacher LLM to a smaller student model for the *Text-to-SQL* task. The central hypothesis is that a formal, structured reasoning signal provides a superior, less ambiguous teaching blueprint than unstructured reasoning. The experiments confirm the hypothesis, demonstrating that distilling structured reasoning is a better teaching method. Detailed error analysis

³BIRD public leaderboard listing is deferred to maintain double-blind anonymity.

indicates that these improvements are primarily due to a significant reduction in syntactic errors. An additional ablation study demonstrated generalization to a different base model. This increased precision, however, comes with a trade-off: the mandatory generation of the intermediate query plan increases the inference token overhead and relies on a fixed reasoning template. Additionally, the Student’s performance remains bound by the Teacher’s fidelity, particularly with respect to complex join and set operations. Future research should therefore explore integrating or combining this structured reasoning with other established strategies to determine the optimal signal for distillation efficiency. Given the demonstrated effectiveness of structured signals in the refinement of syntactic efficiency and logical reasoning, further research is warranted to validate the effect of this structured distillation approach on other complex reasoning tasks beyond *Text-to-SQL*. In summary, the proposed framework facilitates the deployment of high-performing, cost-effective and private models.

6 Acknowledgments

We thank Khalid Eidoo, Abdul Hamid Dabboussi, María Rodríguez-Liñán and Ting Fung Lam for their insightful technical discussions and valuable feedback throughout the project and during the review of this manuscript.

References

- [1] Fei Li and H. V. Jagadish. Constructing an interactive natural language interface for relational databases. *Proc. VLDB Endow.*, 8(1):73–84, 2014.
- [2] Liang Shi, Zhengju Tang, Nan Zhang, Xiaotong Zhang, and Zhi Yang. A survey on employing large language models for text-to-sql tasks. *ACM Comput. Surv.*, 58(2), 2025.
- [3] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *IEEE TKDE*, 2025.
- [4] Cong Duy Vu Hoang, Gioacchino Tangari, Clemence Lanfranchi, Dalu Guo, Paul Cayet, Steve Siu, Don Dharmasiri, Yuan-Fang Li, Long Duong, Damien Hilloulin, Rhicheck Patra, Sungpack Hong, and Hassan Chafi. Distill-C: Enhanced NL2SQL via distilled customization with LLMs. In *NAACL: HLT*. ACL, 2025.
- [5] Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *Advances in Neural Information Processing Systems*, 36, 2024.
- [6] 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. A survey on in-context learning. In *EMNLP*. ACL, 2024.
- [7] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.
- [8] Hanbing Liu, Haoyang Li, Xiaokang Zhang, Ruotong Chen, Haiyong Xu, Tian Tian, Qi Qi, and Jing Zhang. Uncovering the impact of chain-of-thought reasoning for direct preference optimization: Lessons from text-to-sql. *arXiv:2502.11656*, 2025.
- [9] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. Text-to-sql empowered by large language models: A benchmark evaluation. *Proc. VLDB Endow.*, 2024.
- [10] Mohammadreza Pourreza and Davood Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *Neural Information Processing Systems*, 2023.
- [11] Mohammadreza Pourreza, Hailong Li, Ruoxi Sun, Yeounoh Chung, Shayan Talaei, Gaurav Tarlok Kakkar, Yu Gan, Amin Saberi, Fatma Ozcan, and Serkan Ö. Arik. Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql. *Intl. Conf. on Learning Representations*, 2024.
- [12] Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. The death of schema linking? text-to-sql in the age of well-reasoned language models. In *NeurIPS RL Wo*, 2024.
- [13] G Hinton. Distilling the knowledge in a neural network. In *DLRL Workshop, NIPS*, 2014.
- [14] Shiyang Li, Jianshu Chen, Zhiyu Chen, Xinlu Zhang, Zekun Li, Hong Wang, Jing Qian, Baolin Peng, Yi Mao, Wenhui Chen, et al. Explanations from large language models make small reasoners better. In *Sustainable AI*, 2024.
- [15] Cheng-Yu Hsieh, Chun-Liang Li, Chih-kuan Yeh, Hootan Nakhost, Yasuhisa Fujii, Alex Ratner, Ranjay Krishna, Chen-Yu Lee, and Tomas Pfister. Distilling step-by-step! outperforming larger language models with less training data and smaller model sizes. In *Findings ACL*. ACL, 2023.

- [16] Yao Fu, Hao Peng, Litu Ou, Ashish Sabharwal, and Tushar Khot. Specializing smaller language models towards multi-step reasoning. In *Intl. Conf. on Machine Learning*, 2023.
- [17] Gaetano Rossiello, Nhan H Pham, Michael Glass, Junkyu Lee, and Dharmashankar Subramanian. Rationalization models for text-to-SQL. In *Workshop on Reasoning and Planning for LLM*, 2025.
- [18] An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv:2505.09388*, 2025.
- [19] Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. Qlora: Efficient finetuning of quantized llms. *NeurIPS*, 36:10088–10115, 2023.
- [20] Zeyu Han, Chao Gao, Jinyang Liu, Jeff Zhang, and Sai Qian Zhang. Parameter-efficient fine-tuning for large models: A comprehensive survey. *Transactions on ML Research*, 2024.
- [21] Naihao Deng, Yulong Chen, and Yue Zhang. Recent advances in text-to-SQL: A survey of what we have and what we expect. In *Proc. of the 29th Intl. Conf. on Computational Linguistics*, pages 2166–2187, Gyeongju, 2022. Intl. Committee on Computational Linguistics.
- [22] 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. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *EMNLP. ACL*, 2018.
- [23] Xuemei Dong, Chengqi Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. C3: Zero-shot text-to-sql with chatgpt. *arXiv:2307.07306*, 2023.
- [24] Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *arXiv:2204.00498*, 2022.
- [25] Aiwei Liu, Xuming Hu, Lijie Wen, and Philip S Yu. A comprehensive evaluation of chatgpt’s zero-shot text-to-sql capability. *arXiv:2303.13547*, 2023.
- [26] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proc. of the 31st Intl. Conf. on Computational Linguistics. ACL*, 2025.
- [27] Hanchong Zhang, Ruisheng Cao, Lu Chen, Hongshen Xu, and Kai Yu. ACT-SQL: In-context learning for text-to-SQL with automatically-generated chain-of-thought. In *Findings: EMNLP*, 2023.
- [28] Zhishuai Li, Xiang Wang, Jingjing Zhao, Sun Yang, Guoqing Du, Xiaoru Hu, Bin Zhang, Yuxiao Ye, Ziyue Li, Rui Zhao, and Hangyu Mao. Pet-sql: A prompt-enhanced two-round refinement of text-to-sql with cross-consistency. *arXiv:2403.09732*, 2024.
- [29] Xiping Liu and Zhao Tan. Divide and prompt: Chain of thought prompting for text-to-sql. *arXiv:2304.11556*, 2023.
- [30] Mohammadreza Pourreza and Davood Rafiei. DTS-SQL: Decomposed text-to-SQL with small large language models. In *Findings ACL: EMNLP*, pages 8212–8220, Miami, 2024. ACL.
- [31] Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu, Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, and Rasa Hosseinzadeh. MSc-SQL: Multi-sample critiquing small language models for text-to-SQL translation. In *NAACL. ACL*, 2025.
- [32] Bohan Zhai, Canwen Xu, Yuxiong He, and Zhewei Yao. Excot: Optimizing reasoning for text-to-sql with execution feedback. *arXiv:2503.19988*, 2025.
- [33] Charlie Snell, Dan Klein, and Ruiqi Zhong. Learning by distilling context. *arXiv:2209.15189*, 2022.
- [34] Chenglin Li, Qianglong Chen, Liangyue Li, Caiyu Wang, Yicheng Li, Zulong Chen, and Yin Zhang. Mixed distillation helps smaller language model better reasoning. *arXiv:2312.10730*, 2023.
- [35] Kumar Shridhar, Alessandro Stolfo, and Mrinmaya Sachan. Distilling reasoning capabilities into smaller language models. In *Findings ACL*, pages 7059–7073, Toronto, 2023. ACL.