Ada-SQL: Efficient and Adaptive Reasoning for Text-to-SQL

Xiaohan Shen, Xiao Ling, Jie Liu*
College of Artificial Intelligence, Nankai University, Tianjin 300350, CHINA
{shenxiaohan,1120210189}@mail.nankai.edu.cn
jliu@nankai.edu.cn

Abstract

Reasoning efficiency in Text-to-SQL is an important yet underexplored problem. Chain-of-Thought (CoT) reasoning models simplify traditional Text-to-SQL workflows and significantly improve the ability to handle complex queries. However, existing methods typically generate long reasoning processes for all queries. This leads to high token consumption and inference latency, increasing system cost and limiting practical deployment. To address this issue, we propose **Ada-SQL**, an **adaptive reasoning model** for Text-to-SQL that jointly optimizes reasoning mode selection and token budget control via a two-stage reinforcement learning framework coupled with a token budget model. This enables the model to adaptively decide whether to reason and how much reasoning budget to allocate based on query difficulty. Experimental results show that Ada-SQL reduces output token consumption by more than 65% while maintaining competitive execution accuracy, achieving a favorable performance–efficiency trade-off. Code and models will be released at https://github.com/sxhText2SQL/Ada-SQL.

Method	Avg. Latency (s)	Avg. Output Tokens
Greedy		
OmniSQL-7B	1.92	638.37
SQL-R1-7B	2.14	760.56
Self-Consistency (N=8)		
OmniSQL-7B	6.53	5124.40
SQL-R1-7B	7.75	6139.90

Table 1: Average latency and output token consumption of Text-to-SQL reasoning models on BIRD-dev.

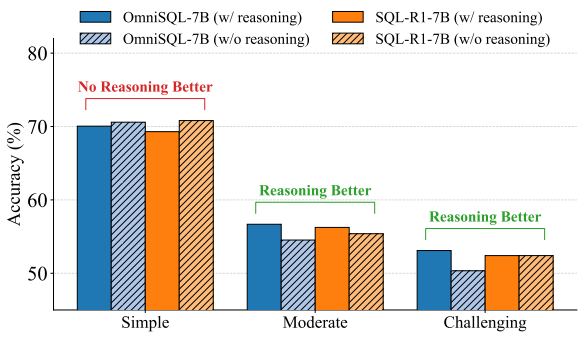


Figure 1: Performance of reasoning vs no reasoning modes across different difficulty levels on BIRD-dev.

1 Introduction

Text-to-SQL translates natural language queries into executable SQL statements. This enables users to retrieve and analyze data without understanding database schemas or SQL syntax. With the rapid development of large language models (LLMs), recent Text-to-SQL systems increasingly rely on complex reasoning and workflow-based paradigms. Methods such as DIN-SQL (Pourreza and Rafiei, 2023) and DeepEye-SQL (Li et al., 2025a) improve complex SQL generation through workflow designs including query decomposition, candidate generation, and multi-stage verification. However, these workflows require multiple LLM calls. As a result, they incur substantial token consumption and inference latency. To improve efficiency, ElieSQL (Zhu et al., 2025) introduces workflow routing for Text-to-SQL by dynamically selecting workflows with different complexity levels. Although it demonstrates the importance of dynamic resource allocation, the method still relies on predefined workflow routing and remains highly coupled with workflow design.

Unlike workflow-based methods, end-to-end reasoning models formulate Text-to-SQL as a step-by-step reasoning process. This simplifies the overall pipeline. Methods such as OmniSQL (Li et al., 2025c), SQL-R1 (Ma et al., 2025), and Reasoning-SQL (Pourreza et al., 2025) introduce Chain-of-Thought (CoT) reasoning and reinforcement learning (RL) optimization, significantly im-

*Corresponding author.

Findings of the Association for Computational Linguistics: EMNLP 2026, pages 32619–32640
October 24-29, 2026 ©2026 Association for Computational Linguistics

proving complex SQL generation while reducing LLM calls. However, existing reasoning models still adopt fixed long-chain reasoning for all queries. This leads to substantial token consumption and inference latency. As shown in Table 1, we evaluate OmniSQL (Li et al., 2025c) and SQL-R1 (Ma et al., 2025) under greedy and self-consistency decoding in terms of inference latency and output tokens. In practical applications, such computational and latency costs not only increase deployment overhead but also degrade user experience, becoming a major bottleneck for large-scale Text-to-SQL systems.

A closer analysis further reveals that not all queries benefit from long-chain reasoning. As illustrated in Figure 1, for simple queries, overly long reasoning chains may introduce noise and cause the model to generate incorrect SQL. This over-reliance on long-chain reasoning may introduce unnecessary computation and occasionally reduce robustness. These observations highlight the need for an *efficient and adaptive* Text-to-SQL reasoning mechanism to make the task truly practical.

To address these challenges, we propose Ada-SQL, an efficient and adaptive Text-to-SQL reasoning model. Unlike existing methods that apply a fixed reasoning strategy to all queries, Ada-SQL jointly optimizes reasoning mode selection and reasoning budget allocation in a unified adaptive framework. Specifically, we introduce a performance-advantage-based efficiency reward that assigns extra rewards to high-performing reasoning behaviors. Based on features of the question, database schema, and generated SQL, the budget model assigns query-specific lower and upper bounds on the number of reasoning tokens. This enables the model to gradually learn an internal resource scheduling policy. Simple queries tend to skip explicit reasoning or use short reasoning chains, whereas complex queries exploit a larger reasoning budget to improve SQL correctness. This adaptive budget mechanism provides fine-grained and continuous control over reasoning resources. It avoids the limitations of discretized difficulty levels and better captures instance-level resource demands. As a result, Ada-SQL achieves a favorable trade-off between performance and efficiency.

In summary, our main contributions are as follows:

- **Ada-SQL: a novel adaptive Text-to-SQL reasoning model.** Ada-SQL adaptively adjusts the reasoning mode and budget for each query, improving inference efficiency and mitigating the resource and latency costs of existing methods.

- **Efficiency-aware optimization for adaptive reasoning.** We design a Text-to-SQL-specific efficiency reward together with token-budget constraints. These components enable the model to learn cost-effective reasoning behaviors under different query complexities.

- **Balanced performance–efficiency trade-off.** Ada-SQL reduces reasoning token consumption while maintaining comparable execution accuracy, achieved through joint optimization of reasoning mode selection and token budget control.

2 Related Work

2.1 Text-to-SQL Workflow and Reasoning Models

Early Text-to-SQL methods mainly relied on workflow-based frameworks (Pourreza and Rafiei, 2023; Gao et al., 2024; Wang et al., 2025; Lee et al., 2025; Xie et al., 2024; Qin et al., 2025; Ling et al., 2024; Pourreza and Rafiei, 2024; Li et al., 2025a). Recent studies have shifted toward end-to-end reasoning models, such as Alpha-SQL (Li et al., 2025b), OmniSQL (Li et al., 2025c), SQL-R1 (Ma et al., 2025), and Arctic-Text2SQL-R1 (Yao et al., 2026). These methods improve complex SQL generation through CoT reasoning, reinforcement learning, or search-based strategies. However, these methods typically rely on MCTS or long-chain reasoning for all queries, leading to high token consumption and inference latency. More detailed discussions are provided in Appendix A.

2.2 Cost-Efficient Text-to-SQL

Most existing studies on Text-to-SQL efficiency focus on SQL execution. However, in practical systems, the cost of generating SQL from user queries can also significantly affect deployment cost and user experience. Ellie-SQL (Zhu et al., 2025) is the first work to highlight this issue. It introduces a workflow-based routing framework that reduces token consumption by about 40% without sacrificing accuracy. However, its router remains tightly coupled with downstream Text-to-SQL components. In contrast, Ada-SQL removes the need for an external router. It internalizes reasoning resource con-

trol into a single model through cold-start training and reinforcement learning.

3 Ada-SQL

3.1 Overview

Ada-SQL aims to generate an executable SQL S with correct execution results for a natural language question Q and database schema D, while minimizing unnecessary reasoning overhead and maintaining strong Text-to-SQL performance. The training pipeline consists of a cold start stage and a reinforcement learning stage. The RL stage is further divided into performance-oriented first-stage RL, construction of a reasoning token budget model, and second-stage RL that explicitly balances performance and efficiency. The overall training pipeline is illustrated in Figure 2.

3.2 Data Preparation

We construct three non-overlapping training sets for cold start, token budget sampling, and reinforcement learning using BIRD-train (Li et al., 2023) and SynSQL-2.5M (Li et al., 2025c). These datasets guide the LLM to learn an adaptive Text-to-SQL reasoning strategy.

Data Construction and Screening For cold start, we construct **SynSQL-ColdStart-16K** for supervised fine-tuning across three reasoning modes (no, short, and long reasoning). Built from SynSQL-2.5M, the dataset contains 16,000 examples spanning 8 question styles and 4 difficulty levels. It includes 5,000 no-reasoning samples, 5,000 short-reasoning samples obtained by compressing the original CoTs, and 6,000 long-reasoning samples. Dataset details and the short-CoT compression procedure are provided in Appendix B.2.

For the token budget model, we build a sampling set of 16,000 non-overlapping examples from SynSQL-2.5M, denoted as **SynSQL-Sampling-16K**, to analyze how reasoning token usage relates to task characteristics and performance (see Appendix B.3).

For reinforcement learning, we train on BIRD-train plus a SynSQL subset with only *Complex* and *Highly complex* examples (2,000 each, 4,000 total), denoted as **SynSQL-RL-4K**, giving 12,326 examples overall. To improve RL efficiency, we filter out samples whose gold-SQL execution result is empty or whose execution time exceeds 30 seconds. Such samples would otherwise mislead learning or slow GRPO training.

Data Splitting For the two-stage RL, we adopt a progressive data usage strategy. In Stage 1, when the model is still unstable, we train only on BIRD-train, which mainly contains relatively simple examples. Once performance improves, Stage 2 uses the combined BIRD-train and SynSQL-RL-4K set to provide a richer difficulty distribution and higher data diversity. Detailed per-stage usage is summarized in Appendix B.4.

3.3 Cold Start

The goal of the cold start stage is to let the model acquire the basic output structure of three reasoning modes. To this end, we perform supervised fine-tuning so that the model can explicitly generate three modes: no reasoning, short reasoning, and long reasoning. The training data comes from the SynSQL-ColdStart-16K dataset.

After cold-start training, the model is standardized to produce both the reasoning process and the final SQL in a unified output format: the reasoning process is enclosed in the `<think>` tag, and the final SQL in the `<answer>` tag. The three reasoning modes differ only in the length of the content within `<think>`. They share the same overall structure. The exact output formats of different modes are shown in Appendix B.2.

3.4 Two-Stage RL & Token Budget Model

In the reinforcement learning stage, we cast training as a Pareto optimization problem over Text-to-SQL performance and reasoning cost. The model should maximize SQL execution accuracy while adaptively selecting the reasoning mode and length to reduce token consumption. This balances performance and efficiency.

We approximate the Pareto frontier with a two-stage scalarization strategy. Stage 1 optimizes only for performance. This allows the model to approach an accuracy-oriented policy under almost no constraint on reasoning length. Stage 2 adds an efficiency reward in addition to the performance reward. This imposes resource constraints on the Stage 1 policy and pushes it toward a more balanced performance–efficiency solution.

We use Group Relative Policy Optimization (GRPO) (Shao et al., 2024) for RL training. For each input query, the policy model generates a group of outputs $G = \{o_1, o_2, \ldots, o_i\}$. Each output o_i receives a reward r_i according to the stage-specific reward function, from which an advantage

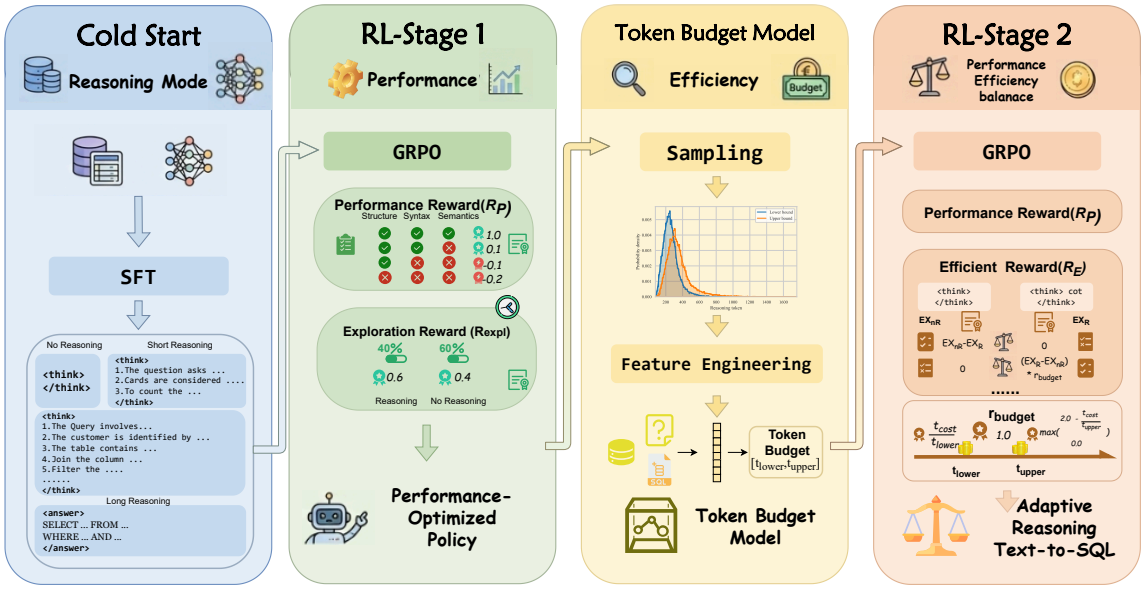


Figure 2: Overview of the Ada-SQL training pipeline.

$\hat{A}_i$ is computed, and the policy is updated by optimizing the objective in Equation 1.

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|o_i|}\sum_{t=1}^{|o_i|}\left(\min\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}\hat{A}_i,\ \text{clip}\left(\frac{\pi_\theta(o_i|q)}{\pi_{\theta_{old}}(o_i|q)}, 1-\epsilon, 1+\epsilon\right)\hat{A}_i\right) - \beta\,\mathbb{D}_{\text{KL}}\left(\pi_\theta \,\|\, \pi_{\text{ref}}\right)\right)\right] \quad (1)$$

3.4.1 RL Stage 1: Performance-Oriented

In the first stage, we optimize solely for SQL result correctness, aiming to obtain a stable and high-performing policy from the cold start model. This provides a reliable initialization before introducing efficiency constraints.

Concretely, each output is checked at three levels: structure, syntax, and semantics. The structural check enforces the output format; the syntactic check requires the generated SQL to execute successfully on the database engine; and the semantic check requires its execution result to match that of the gold SQL. Based on these checks, we define the performance reward R_P as:

$$R_P = \begin{cases} 1.0, & \text{all correct} \\ 0.1, & \text{semantics are incorrect} \\ -0.1, & \text{syntax \& semantics are incorrect} \\ -0.2, & \text{all incorrect} \end{cases} \quad (2)$$

Since the initial policy supports multiple reasoning modes, relying only on the performance reward tends to collapse into a single mode in early training. To encourage exploration, we define an exploration reward R_{expl} separately for the "reasoning" and "no reasoning" behaviors. Let $Proportion$ denote the fraction of a given behavior in the current candidate set G, then

$$R_{expl} = 1 - Proportion \quad (3)$$

The exploration reward is activated only in early training with a linearly decayed weight. It is granted solely to outputs that already achieve the full performance reward ($R_P = 1$), so as to preserve mode diversity without changing the main objective. Overall, the total reward in Stage 1 is:

$$R_{\text{Stage1}} = R_P + \alpha_{expl} R_{expl} \quad (4)$$

3.4.2 Text-to-SQL Token Budget Model

After Stage 1 RL with a performance-oriented objective, the model can generate Text-to-SQL queries well, but does not yet account for reasoning costs or token allocation. We estimate per-

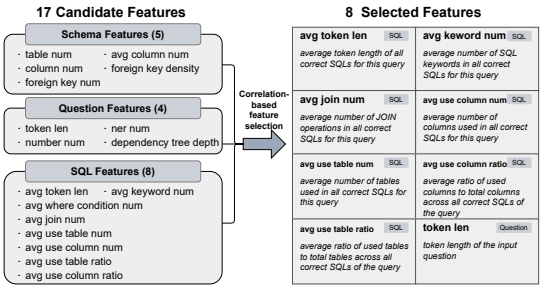


Figure 3: Feature engineering for the Text-to-SQL.

query token budget intervals using large-scale sampling on SynSQL-Sampling-16K; details are in Appendix C.1.

To capture key factors behind reasoning cost, we construct 17 Text-to-SQL features from the schema, question, and SQL perspectives, and measure their relevance to the lower and upper bounds of reasoning tokens. We then select 8 features to estimate the token budget, as shown in Figure 3. Remaining feature definitions and full correlation statistics are given in Appendices C.2, C.3, and C.4.

We model the token budget interval with a LightGBM-based (Ke et al., 2017) quantile regression model. Two regressors predict the lower and upper bounds of token usage, yielding a budget interval $[t_l, t_u]$ from the 8 selected features by minimizing the pinball loss. Compared with mean-regression models, this approach better captures distributional asymmetry and heteroscedasticity. It is also more robust to long-tailed and outlier cases and directly provides interval boundaries for resource allocation. The learned token budget model is used in RL Stage 2 as a constraint on reasoning cost to guide the model's reasoning behavior.

3.4.3 RL Stage 2: Balancing Performance and Efficiency

In Stage 2, we add an efficiency reward based on the Stage 1 performance reward to encourage controlling reasoning cost while preserving SQL correctness. For each example, we split outputs G into G_R (reasoning) and G_{nR} (no reasoning) and compute their execution accuracies EX_R and EX_{nR}. An efficiency reward is given only when exactly one group reaches accuracy 1; otherwise it is zero for both. We call this a performance-advantage-based efficiency reward. For the no reasoning group, it equals the accuracy gap between the two groups. For the reasoning group, it equals this gap multiplied by the budget satisfaction ratio r_{budget}.

The budget satisfaction ratio is given by the token budget model. Let t_{cost} denote the token cost of the reasoning chain in an output and $[t_l, t_u]$ the budget interval, then

$$r_{budget} = \begin{cases} 1, & t_l \le t_{cost} \le t_u \\ \frac{t_{cost}}{t_l}, & t_{cost} < t_l \\ \max\left(2 - \frac{t_{cost}}{t_u}, 0\right), & t_{cost} > t_u \end{cases} \tag{5}$$

The performance-advantage-based efficiency reward R_E is defined in Table 2.

R_E		EX	
R_E^R	R_E^{nR}	EX_R	EX_{nR}
$(EX_R - EX_{nR}) \times r_{budget}$	0	=1	<1
0	$EX_{nR} - EX_R$	<1	=1
0	0	<1	<1
0	0	=1	=1

Table 2: Efficiency reward computation based on performance advantage.

This efficiency reward is added on top of the performance reward. It is activated only when the output passes the structural, syntactic, and semantic checks (i.e., $R_P = 1$).

Overall, the total reward in Stage 2 is:

$$R_{\text{Stage2}} = \begin{cases} R_P + R_E, & R_P = 1 \\ R_P, & R_P < 1 \end{cases} \tag{6}$$

4 Experiments

4.1 Experimental Setup

Training Setup. We take OmniSQL-7B (Li et al., 2025c) as the base model and sequentially apply cold start and RL, both with LoRA (Hu et al., 2022) adaptation. Full training configurations are given in Appendix D.1.

Benchmark. Experiments are conducted on the BIRD-dev, one of the most widely used Text-to-SQL benchmarks. BIRD-dev contains 1,534 examples, with queries labeled as *simple*, *moderate*, or *challenging*. Generalization analysis is mainly conducted on Spider (Yu et al., 2018).

Metrics. We evaluate both performance and efficiency. Performance is measured by Execution Accuracy (EX), which indicates whether the generated SQL yields the same execution result as the gold SQL. Efficiency is measured by token consumption and average inference latency per example.

Baselines. We compare Ada-SQL with three types of baselines: (i) open-source reasoning models, including Qwen3-8B (Team, 2025), OmniSQL-7B (Li et al., 2025c), and SQL-R1-7B (Ma et al., 2025); (ii) EllieSQL (Zhu et al., 2025), a cost-efficient Text-to-SQL method; and (iii) workflow-based methods, such as MAC-SQL (Wang et al., 2025), TA-SQL (Qu et al., 2024), RSL-SQL (Cao et al., 2024), CHESS (Talaei et al., 2024), and AutoLink (Wang et al., 2026).

Environment. To ensure fair comparison, all experiments share the same hardware and inference settings. Detailed configurations and prompts are provided in Appendix D.2 and D.3. The token budget model is only used during Stage 2 RL training and is not involved during inference.

4.2 Main Results

Comparison with Reasoning Model Baselines. Table 3 compares Ada-SQL with existing Text-to-SQL reasoning models. Under greedy decoding, Ada-SQL maintains comparable accuracy while reducing average latency and output tokens by about 31% and 65% compared with the average of SQL-R1 and OmniSQL. Its reasoning token range (0–629) shows that the model can skip or shorten reasoning for simple queries.

Under self-consistency, Ada-SQL achieves 66.36% EX while still requiring substantially fewer tokens and lower latency than SQL-R1 and OmniSQL. Overall, the results demonstrate that adaptive reasoning control can effectively improve the performance–efficiency trade-off in Text-to-SQL. We additionally compare Ada-SQL with OmniSQL on BIRD Mini-Dev, a high-quality subset of BIRD, with results reported in Appendix D.4.2.

Comparison with EllieSQL. Table 4 compares Ada-SQL with EllieSQL, a workflow-based efficient Text-to-SQL framework built on GPT-4o-mini-2024-07-18. Unlike EllieSQL, Ada-SQL performs adaptive reasoning within a single 7B model instead of routing queries across predefined workflows. Following EllieSQL, we measure token cost using:

$$WeightToken = Token_{input} + 4 \times Token_{output} \quad (7)$$

denoted as WT. Ada-SQL improves EX from 55.41% to 64.54% under greedy decoding and reaches 66.36% with self-consistency. Meanwhile, it reduces WT by 58.7% under greedy decoding. Although self-consistency increases token usage due to multiple sampling, Ada-SQL still achieves a favorable efficiency–performance trade-off.

Comparison with Methods based on Closed-Source LLM Workflows. Compared with workflow-based systems that rely on closed-source LLMs and multi-stage pipelines, Ada-SQL performs adaptive reasoning within a single 7B model. It does not require external workflow scheduling. As shown in Table 5, Ada-SQL achieves comparable performance with substantially lower token consumption.

Generalization capability of Ada-SQL. To evaluate the generalization capability of our method, we conduct experiments on the Spider dataset, with results reported in Appendix D.4.1. Without training on Spider, Ada-SQL still achieves a substantial reduction in token consumption, demonstrating its generalization ability in improving efficiency.

4.3 Ablation Study

Impact of the Staged Training Strategy. As shown in Table 6, with only the cold start training, the EX is 59.13% with an average of 118.32 output tokens. After introducing Stage 1 RL, EX improves, but the average number of output tokens increases substantially. This indicates that the performance reward based on execution correctness can improve Text-to-SQL quality, but also encourages the model to generate longer reasoning chains. After further introducing Stage 2 RL, EX continues to improve to 64.54%, while the number of output tokens decreases compared with Stage 1. These results demonstrate that the staged optimization with efficiency constraints effectively suppresses excessive growth of reasoning token length while continuously improving performance.

Cold Start	RL		EX (%)	Tokens
	Stage1	Stage2		
✓	✗	✗	59.13	118.32
✓	✓	✗	62.32	270.73
✓	✓	✓	64.54	242.18

Table 6: Ablation of the staged training strategy.

Impact of the Efficiency Reward. As shown in Table 7, with only the performance reward R_P, the model reaches an EX of 64.15%, but the average number of output tokens increases to 387.66, showing a strong reliance on long reasoning chains with-

Method	Size	Performance	Efficiency			
		EX (%)	Avg. Latency (s)	Avg. Output Tokens	Reasoning Tokens (Max)	Reasoning Tokens (Min)
Greedy						
Qwen3	8B	50.46	3.35	1062.54	2047	174
OmniSQL	7B	64.41	1.92	638.37	1019	213
SQL-R1	7B	63.75	2.14	760.56	2047	323
Ada-SQL	7B	**64.54**	**1.39** (↓ 31.53%)	**242.18** (↓ 65.38%)	**629** (↓ 58.97%)	**0** (↓ 100.00%)
Self-Consistency (N=8)						
Qwen3	8B	51.56	27.20	8567.86	16370	1697
OmniSQL	7B	65.78	6.53	5124.40	7255	2102
SQL-R1	7B	65.78	7.75	6139.90	11467	2990
Ada-SQL	7B	**66.36**	**3.03** (↓ 57.56%)	**1857.16** (↓ 67.03%)	**3904** (↓ 58.30%)	**0** (↓ 100.00%)

Table 3: Comparison of Ada-SQL with existing Text-to-SQL reasoning models on BIRD-dev. EX denotes execution accuracy. Efficiency metrics report average latency per example, average output tokens, and reasoning token ranges. Green numbers indicate relative reductions compared to the average of OmniSQL and SQL-R1. Results are shown for both Greedy and Self-Consistency (N=8) decoding strategies.

Method	EX	Avg. WT
EllieSQL	55.41	7641.51
Ada-SQL(Gre)	64.54	**3155.76** (↓ 58.7%)
Ada-SQL(SC N=8)	**66.36**	9615.68 (↑ 25.8%)

Table 4: Comparison with the routing-based efficient Text-to-SQL system EllieSQL on BIRD-dev. EllieSQL results are taken from the original paper. "SC" indicates Self-Consistency.

out efficiency constraints. Adding an efficiency reward without budget constraints, R_E(w/o Budget), reduces output tokens to 189.34 but also lowers EX to 61.86%, partly due to an implicit length bias in GRPO: averaging the loss over sequence length gives shorter responses larger per-token gradients under positive reward, which may over-compress reasoning on complex queries.

Method	EX(%)	Tokens
R_P	64.15	387.66
$R_P + R_E(w/o\ Budget)$	61.86	189.34
$R_P + R_E$	64.54	242.18

Table 7: Ablation of the efficiency reward design.

After introducing the token budget model, the efficiency reward can constrain reasoning length according to task characteristics, leading to more appropriate allocation of reasoning tokens. Combining the performance reward with the budget-aware efficiency reward ($R_P + R_E$) yields a better trade-off between EX and token cost, substantially shortening reasoning while slightly improving the accuracy of R_P alone.

4.4 Analysis of Token Budget Mechanism

Generalization of the Token Budget Model. We further evaluate the generalization ability of the token budget model on Qwen3-1.7B and Qwen3-4B, as shown in Table 8.

Method	EX (%)	Avg. Latency (s)	Avg. Output Tokens
1.7B			
Qwen3	33.18	0.82	707.73
Ada-SQL	**38.33**	**0.44** (↓ 46.34%)	**137.50** (↓ 80.57%)
4B			
Qwen3	48.04	2.36	1057.57
Ada-SQL	**55.35**	**1.13** (↓ 52.12%)	**250.44** (↓ 76.32%)

Table 8: Generalization of the token budget mechanism across different backbone model sizes (Qwen3-1.7B and Qwen3-4B).

Although the budget model is not trained or sampled on these model sizes, Ada-SQL consistently

Method	Model(Gen)	Performance	Avg. Token Consumption (Input+Output)		
		EX(%)	Schema Linking	SQL Generation	Total
MAC-SQL	GPT-4	59.39	-	6800	$\geqslant$ 6800
TA-SQL	GPT-4	56.19	-	7300	$\geqslant$ 7300
RSL-SQL	GPT-4o	67.21	14800	7500	22300
CHESS	Gemini-1.5-Pro	68.31	-	14500	$\geqslant$ 14500
AutoLink	Deepseek-V3	66.36	-	-	8000
AutoLink	Gemini-1.5-Pro	**68.71**	-	-	8000
Ada-SQL(7B) Greedy		64.54	-	-	**2429.22**
Ada-SQL(7B) SC(N=8)		66.36	-	-	4044.20

Table 5: Comparison with Text-to-SQL methods based on closed-source LLM workflows. EX and token statistics for baselines are taken from AutoLink. Workflow-based methods typically include schema linking and SQL generation; "Model(Gen)" denotes the LLM used for SQL generation, and schema linking in AutoLink is consistently performed by DeepSeek-V3. "–" indicates that token usage is not reported in AutoLink; for these baselines, we report only generation tokens and mark them with $\geqslant$ to reflect possible extra cost from schema linking. Following AutoLink, token consumption is measured as input + output tokens. "SC" denotes Self-Consistency.

Budget Supervision	EX (%)	Avg. Output Tokens
FP (10%–50%)	60.95	72.78
FP (10%–75%)	61.73	187.41
Ada-SQL	64.54	242.18

Table 9: Comparison with simpler budget intervals on the BIRD dev set. Fixed Percentile (FP) denotes a fixed budget interval whose lower and upper bounds are determined by the corresponding percentiles of reasoning-token lengths in rollout outputs. For example, FP (10%–50%) uses the 10th and 50th percentiles as its lower and upper bounds, respectively.

reduces inference latency and output token consumption while maintaining competitive execution accuracy. These results indicate that the learned token budget mechanism generalizes effectively across different model scales, demonstrating strong transferability to models of varying sizes.

Comparison with Simpler Budget Mechanisms. We compare the budget mechanism of Ada-SQL with simpler alternatives. We keep the entire training pipeline unchanged and replace only the budget supervision in the stage 2 RL with fixed budget intervals derived from empirical percentiles of the reasoning-token lengths of rollout outputs (e.g., 10%–50% and 10%–75%). This heuristic represents a simple approach to constraining reasoning length using predefined budget rules.

As shown in Table 9, percentile-based supervision does reduce token consumption further, and tightening the budget from the 75th to the 50th percentile shortens reasoning even more, but execution accuracy drops markedly in both cases. A key limitation lies in the source of the budget signal: empirical percentiles reflect the model's previous outputs rather than the intrinsic reasoning requirements of each query. As training increasingly favors shorter trajectories, the resulting supervision may further reinforce this tendency, creating a self-reinforcing process toward little or no reasoning. Consequently, such supervision can impair performance on complex queries that require longer reasoning trajectories.

4.5 Analysis of Adaptive Reasoning Behavior

To analyze the reasoning behavior of different methods, we plot the distribution of reasoning tokens for each model in Figure 4. SQL-R1 and OmniSQL tend to produce long reasoning chains for most examples. In contrast, Ada-SQL is skewed toward low token usage and shows a clear probability mass at 0 tokens. This indicates that it can directly generate SQL without explicit reasoning for a subset of examples.

To further analyze the adaptive behavior of Ada-SQL, we examine reasoning usage across difficulty levels in Figure 5. For *simple* queries, about 22.49% of examples generate SQL without reasoning, while most others use short reasoning (< 300 tokens). *Moderate* queries also mainly rely on short reasoning. As difficulty increases, the proportion of long reasoning rises significantly on *challenging* queries. A similar trend is also observed in the cross-dataset analysis on Spider. This further confirms the robustness of the adaptive behavior across datasets (see Appendix D.4.1 for details). This

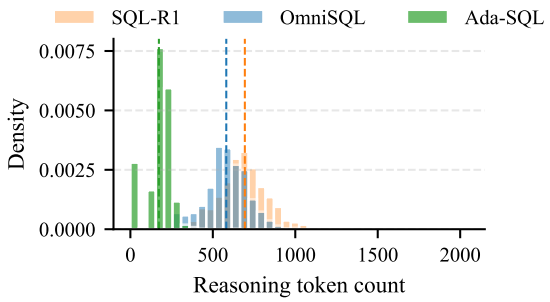


Figure 4: Distributions of reasoning token usage for Ada-SQL, SQL-R1, and OmniSQL. The dashed line indicates the mean reasoning token consumption.

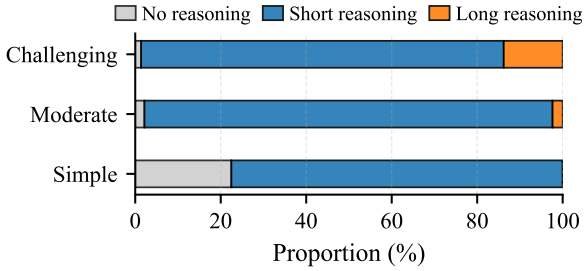


Figure 5: Distribution of reasoning modes across different difficulty levels.

shows that Ada-SQL reduces unnecessary reasoning on easy queries while allocating more budget to harder ones.

We further analyze budget adherence during training and inference. The adherence ratio steadily increases during training and remains high at inference time (90.49% satisfaction without explicit budget signals), as detailed in Appendix D.5. These results indicate that the budget signal is effectively internalized, enabling implicit control of reasoning length.

Overall, Ada-SQL learns a *difficulty-aware* reasoning strategy. It allocates minimal reasoning for simple queries and progressively increases reasoning depth for harder ones. This reduces overall inference cost while preserving performance. Representative cases are provided in Appendix E.

5 Conclusion

We present Ada-SQL, an efficient adaptive Text-to-SQL reasoning model. By combining cold start training with a two-stage reinforcement learning framework and a token budget model derived from Text-to-SQL task characteristics, Ada-SQL adaptively adjusts both reasoning mode and depth for each query. Experiments show that Ada-SQL improves the efficiency–performance trade-off by reducing reasoning overhead. Future work will explore finer-grained control of reasoning resources for more complex Text-to-SQL scenarios.

Limitations

Ada-SQL is primarily evaluated on a 7B-parameter model, which serves as the main setting and is representative of low-resource and privacy-sensitive deployment scenarios in Text-to-SQL applications. Experiments on 1.7B and 4B models further validate the generalization of our method across model scales. However, we have not extensively studied its behavior with larger language models. It remains an open question whether larger models, with stronger inherent reasoning capabilities, would reduce the need for explicit adaptive control or exhibit different sensitivities to token budget constraints.

In addition, although Ada-SQL uses general Text-to-SQL task features that are independent of specific datasets and models for token budget estimation, the current budget modeling process still relies on manually designed features rather than fully end-to-end budget allocation from the query input. Exploring more dynamic and end-to-end budget modeling strategies, such as query-aware or dynamically learned budget allocation, remains an important direction for efficient Text-to-SQL research.

Acknowledgements

This research is supported by the National Key Research and Development Program of China under the grant No. 2023YFF0725003, National Natural Science Foundation of China under the grant No. 62376129, Tianjin Science and Technology Plan Project under the grant No. 25ZGZNGX00100, Hebei Province Science and Technology Plan Project under the grant No. 26241703D, Tianjin Science and Technology Plan Project under the grant No. 24JCYBJC01950, the Fundamental Research Funds for the Central Universities under the grant No. 63261005, and Nankai University-AlphaESS Smart Energy Storage Joint Research Center Project.

References

Zhenbiao Cao, Yuanlei Zheng, Zhihao Fan, Xiaojin Zhang, Wei Chen, and Xiang Bai. 2024. RSL-SQL: robust schema linking in text-to-sql generation. *CoRR*, abs/2411.00073.

DeepSeek-AI. 2024. Deepseek-v3 technical report. *CoRR*, abs/2412.19437.

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

Mingqian He, Yongliang Shen, Wenqi Zhang, Qiuying Peng, Jun Wang, and Weiming Lu. 2025. Star-sql: Self-taught reasoner for text-to-sql. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 24365–24375. Association for Computational Linguistics.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. 2022. Lora: Low-rank adaptation of large language models. In *The Tenth International Conference on Learning Representations, ICLR 2022, Virtual Event, April 25-29, 2022*. OpenReview.net.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. Lightgbm: A highly efficient gradient boosting decision tree. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017, December 4-9, 2017, Long Beach, CA, USA*, pages 3146–3154.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph Gonzalez, Hao Zhang, and Ion Stoica. 2023. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the 29th Symposium on Operating Systems Principles, SOSP 2023, Koblenz, Germany, October 23-26, 2023*, pages 611–626. ACM.

Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2025. MCS-SQL: leveraging multiple prompts and multiple-choice selection for text-to-sql generation. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 337–353. Association for Computational Linguistics.

Boyan Li, Chong Chen, Zhujun Xue, Yinan Mei, and Yuyu Luo. 2025a. Deepeye-sql: A software-engineering-inspired text-to-sql framework. *CoRR*, abs/2510.17586.

Boyan Li, Jiayi Zhang, Ju Fan, Yanwei Xu, Chong Chen, Nan Tang, and Yuyu Luo. 2025b. Alpha-sql: Zero-shot text-to-sql using monte carlo tree search. In *Forty-second International Conference on Machine Learning, ICML 2025, Vancouver, BC, Canada, July 13-19, 2025*, Proceedings of Machine Learning Research. PMLR / OpenReview.net.

Haoyang Li, Shang Wu, Xiaokang Zhang, Xinmei Huang, Jing Zhang, Fuxin Jiang, Shuai Wang, Tieying Zhang, Jianjun Chen, Rui Shi, Hong Chen, and Cuiping Li. 2025c. Omnisql: Synthesizing high-quality text-to-sql data at scale. *Proc. VLDB Endow.*, 18(11):4695–4709.

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Chenhao Ma, Guoliang Li, Kevin Chen-Chuan Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM already serve as A database interface? A big bench for large-scale database grounded text-to-sqls. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Xiao Ling, Jialin Liu, Jindu Liu, Jianhua Wu, and Jie Liu. 2024. Finetuning llms for text-to-sql with two-stage progressive learning. In *Natural Language Processing and Chinese Computing - 13th National CCF Conference, NLPCC 2024, Hangzhou, China, November 1-3, 2024, Proceedings, Part II*, Lecture Notes in Computer Science, pages 449–461. Springer.

Peixian Ma, Xialie Zhuang, Chengjin Xu, Xuhui Jiang, Ran Chen, and Jian Guo. 2025. SQL-R1: training natural language to SQL reasoning model by reinforcement learning. *CoRR*, abs/2504.08600.

Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, Jake VanderPlas, Alexandre Passos, David Cournapeau, Matthieu Brucher, Matthieu Perrot, and Edouard Duchesnay. 2011. Scikit-learn: Machine learning in python. *J. Mach. Learn. Res.*, 12:2825–2830.

Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: decomposed in-context learning of text-to-sql with self-correction. In *Advances in Neural Information Processing Systems 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: decomposed text-to-sql with small large language models. In *Findings of the Association for Computational Linguistics: EMNLP 2024, Miami, Florida, USA, November 12-16, 2024*, Findings of ACL, pages 8212–8220. Association for Computational Linguistics.

Mohammadreza Pourreza, Shayan Talaei, Ruoxi Sun, Xingchen Wan, Hailong Li, Azalia Mirhoseini, Amin Saberi, and Sercan Ö. Arik. 2025. Reasoning-sql: Reinforcement learning with SQL tailored partial rewards for reasoning-enhanced text-to-sql. *CoRR*, abs/2503.23157.

Yang Qin, Chao Chen, Zhihang Fu, Ze Chen, Dezhong Peng, Peng Hu, and Jieping Ye. 2025. ROUTE: robust multitask tuning and collaboration for text-to-sql.

In *The Thirteenth International Conference on Learning Representations, ICLR 2025, Singapore, April 24-28, 2025*. OpenReview.net.

Ge Qu, Jinyang Li, Bowen Li, Bowen Qin, Nan Huo, Chenhao Ma, and Reynold Cheng. 2024. Before generation, align it! A novel and effective strategy for mitigating hallucinations in text-to-sql generation. In *Findings of the Association for Computational Linguistics, ACL 2024, Bangkok, Thailand and virtual meeting, August 11-16, 2024*, Findings of ACL, pages 5456–5471. Association for Computational Linguistics.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: contextual harnessing for efficient SQL synthesis. *CoRR*, abs/2405.16755.

Qwen Team. 2025. Qwen3 technical report. *CoRR*, abs/2505.09388.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025. MAC-SQL: A multi-agent collaborative framework for text-to-sql. In *Proceedings of the 31st International Conference on Computational Linguistics, COLING 2025, Abu Dhabi, UAE, January 19-24, 2025*, pages 540–557. Association for Computational Linguistics.

Ziyang Wang, Yuanlei Zheng, Zhenbiao Cao, Xiaojin Zhang, Zhongyu Wei, Pei Fu, Zhenbo Luo, Wei Chen, and Xiang Bai. 2026. Autolink: Autonomous schema exploration and expansion for scalable schema linking in text-to-sql at scale. In *Fortieth AAAI Conference on Artificial Intelligence, Thirty-Eighth Conference on Innovative Applications of Artificial Intelligence, Sixteenth Symposium on Educational Advances in Artificial Intelligence, AAAI 2026, Singapore, January 20-27, 2026*, pages 33809–33817. AAAI Press.

Yuanzhen Xie, Xinzhou Jin, Tao Xie, Mingxiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, Chengxiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for enhancing attention: Improving LLM-based text-to-SQL through workflow paradigm. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 10796–10816, Bangkok, Thailand. Association for Computational Linguistics.

Zhewei Yao, Guoheng Sun, Lukasz Borchmann, Zheyu Shen, Minghang Deng, Bohan Zhai, Hao Zhang, Ang Li, and Yuxiong He. 2026. Arctic-text2sql-r1: Simple rewards, strong reasoning in text-to-sql. In *Findings of the Association for Computational Linguistics, ACL 2026, San Diego, California, United States, July 2-7, 2026*, pages 26966–26995. 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 R. Radev. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Yizhang Zhu, Runzhi Jiang, Boyan Li, Nan Tang, and Yuyu Luo. 2025. Elliesql: Cost-efficient text-to-sql with complexity-aware routing. *CoRR*, abs/2503.22402.

A Detailed Discussion of Related Work

A.1 Methods based on Workflow

Workflow-based methods are one of the most widely used paradigms in Text-to-SQL and have inspired many representative studies:

- **DIN-SQL** (Pourreza and Rafiei, 2023) decomposes Text-to-SQL into schema linking, query classification and decomposition, SQL generation, and self-correction modules, achieving strong performance.
- **DAIL-SQL** (Gao et al., 2024) systematically studies prompt engineering and example selection strategies for Text-to-SQL.
- **MAC-SQL** (Wang et al., 2025) introduces three collaborative agents, including a selector, decomposer, and refiner, to improve SQL generation.
- **MCS-SQL** (Lee et al., 2025) explores a broader solution space through multiple prompts and aggregates candidates across schema linking, SQL generation, and candidate selection stages.
- **DeepEye-SQL** (Li et al., 2025a) formulates Text-to-SQL as a multi-stage workflow and optimizes it with multiple generators and toolchains.
- **DTS-SQL** (Pourreza and Rafiei, 2024) fine-tunes specialized models for schema linking and SQL generation, achieving performance comparable to large-scale models with a 7B model.

- **ROUTE** (Qin et al., 2025) decomposes the task into subtasks such as Text-to-SQL, schema linking, noise correction, and continuation, and progressively generates SQL through multi-task tuning and collaborative prompting.

These methods improve Text-to-SQL performance through prompt engineering and task decomposition, enabling LLMs to collaborate across multiple subtasks. However, they usually require multiple LLM calls, resulting in high inference cost and latency.

A.2 Methods based on End-to-End Text-to-SQL

With the development of Chain-of-Thought fine-tuning and reinforcement learning, end-to-end Text-to-SQL methods have gradually become the dominant paradigm. Representative directions include MCTS-based methods and Step-by-Step reasoning models:

- **Alpha-SQL** (Li et al., 2025b) uses MCTS to explore richer solution spaces and improve SQL generation quality.
- **STaR-SQL** (He et al., 2025) enables models to learn step-by-step Text-to-SQL reasoning through iterative fine-tuning.
- **OmniSQL** (Li et al., 2025c) constructs the large-scale SynSQL-2.5M dataset with reasoning annotations and trains a high-performance reasoning model.
- **SQL-R1** (Ma et al., 2025) designs progressive reward mechanisms based on GRPO reinforcement learning to encourage reasoning behavior.
- **Reasoning-SQL** (Pourreza et al., 2025) introduces diversified reward functions and incorporates LLM-based judging into the reward framework.
- **Arctic-Text2SQL-R1** (Yao et al., 2026) achieves state-of-the-art performance with a simple reward design.

Compared with workflow-based approaches, these methods avoid repeated LLM calls but still suffer from high reasoning cost. MCTS-based methods require multi-path exploration, while Step-by-Step reasoning models typically apply long-chain reasoning to all queries, leading to substantial token consumption and inference latency.

A.3 Ada-SQL: Single-pass Adaptive Reasoning Paradigm

Compared with workflow-based methods, Ada-SQL requires only a single LLM call, thereby reducing the cost and latency introduced by multi-call pipelines. Compared with existing end-to-end Text-to-SQL methods, Ada-SQL eliminates both search procedures and fixed long-chain reasoning, and instead adopts a deployment-oriented adaptive reasoning paradigm with budget awareness and dynamic mode switching, effectively optimizing inference cost.

B Training Data Construction and Composition

B.1 SynSQL-2.5M

SynSQL-2.5M (Li et al., 2025c) consists of 2.5 million examples, each containing the basic information shown in Table 10.

B.2 SynSQL-ColdStart-16K

In the cold start stage, we construct training data that jointly covers three output formats: no reasoning, short reasoning, and long reasoning. To this end, we use SynSQL-2.5M with CoT annotations as the source corpus and select 16,000 examples, each from a distinct database. These examples span 8 question styles (excluding *Conversational*) and 4 complexity levels. The overall composition is summarized in Table 11.

Complexity	Number of Examples per style	Reasoning mode	Number of Examples
Simple	625	No reasoning	5,000
Moderate	625	Short reasoning	5,000
Complex	375	Long reasoning	3,000
Highly Complex	375	Long reasoning	3,000

Table 11: Composition of the SynSQL-ColdStart-16K dataset by complexity level and reasoning mode.

Field	Description
db_id	Identifier of the underlying database
sql_complexity	SQL complexity level: *Simple*, *Moderate*, *Complex*, *Highly Complex*
question_style	Question style: *Formal*, *Colloquial*, *Imperative*, *Interrogative*, *Descriptive*, *Concise*, *Vague*, *Metaphorical*, *Conversational*
question	Natural language question
external_knowledge	Relevant auxiliary knowledge
cot	Reasoning process leading to the gold SQL
sql	Gold SQL

Table 10: Fields contained in each example of SynSQL-2.5M.

Reasoning Mode	Output Format
No Reasoning	<think></think> <answer>```sql SQL```</answer>
Short Reasoning	<think>short reasoning</think> <answer>```sql SQL```</answer>
Long Reasoning	<think>long reasoning</think> <answer>```sql SQL```</answer>

Table 12: Structured output formats for the three reasoning modes used in Ada-SQL.

For long reasoning examples, we directly use the CoT annotations in SynSQL-2.5M as the reasoning process. For short reasoning examples, we design the compression prompt shown in Figure 6 and employ DeepSeek-V3.2 (DeepSeek-AI, 2024) to compress the original CoT into short reasoning labels.

After cold-start training on SynSQL-ColdStart-16K, the model is trained to generate both intermediate reasoning and the final SQL in a unified format. The reasoning process is enclosed within the <think> tag, while the final SQL is enclosed within the <answer> tag. Different reasoning modes are distinguished only by the length of the content within the <think> tag. The output formats are shown in Table 12.

B.3 SynSQL-Sampling-16K

To systematically analyze the relationship between reasoning cost, Text-to-SQL task characteristics, and model performance, we construct the SynSQL-Sampling-16K dataset and perform large-scale generation sampling on it to obtain realistic and diverse analysis data.

Similar to SynSQL-ColdStart-16K, SynSQL-Sampling-16K is randomly sampled from SynSQL-2.5M. To maximize data diversity while maintaining the overall scale, we uniformly sample over the combinations of 4 difficulty levels and 8 question styles, selecting 500 examples for each combination, resulting in 16,000 examples in total. There is no sample overlap between SynSQL-Sampling-16K and SynSQL-ColdStart-16K, nor with the datasets used for reinforcement learning training.

B.4 Data Split

In Section 3.2, we briefly introduced the data partitioning strategy used for two-stage reinforcement learning: in the first stage, where model performance remains unstable, we train only on BIRD-train, which contains relatively easy examples; once the performance has been substantially improved in the first stage, the second stage adopts a combined dataset of BIRD-train and SynSQL-RL-4K to jointly account for both difficulty distribution and data diversity.

The data sources, scales, and compositions used in the cold start stage, the two-stage reinforcement learning training, and the reasoning token budget sampling stage are summarized in Table 13.

C Relationship Between Text-to-SQL Task Characteristics and Reasoning Token Budgets

C.1 Computing the Token Budget

We use the model trained in the first stage reinforcement learning phase to perform sampling on SynSQL-Sampling-16K, generating 32 outputs for each example. We then apply the following procedure to derive the lower and upper token budget bounds for each example.

Stage	Dataset	Number of Examples	CoT Labels
Cold Start	SynSQL-ColdStart-16K	16000	✓(No/ Short/Long)
Token Budget Sampling	SynSQL-Sampling-16K	16000	✗
RL-Stage1	BIRD-train	8326	✗
RL-Stgae2	BIRD-train + SynSQL-RL-4K	12326	✗

Table 13: Data splits used in different stages of Ada-SQL training.

Prompt for compressing CoT

You are an assistant for Text-to-SQL reasoning process compression. Please compress the chain-of-thought reasoning process (cot) used to derive the gold SQL, based on the given database schema, question, and external knowledge.
Requirements:
1. Retain all critical reasoning steps and logical transition points
2. Remove repetitive, redundant, or obvious reasoning steps
3. Maintain the complete core logical chain of SQL generation
4. Rephrase using more concise language without losing key information
5. The overall format should follow the original cot format. For example, the final SQL should be preceded by phrases like "Here's how the query can be written:".
Start directly with the compressed reasoning steps without any preliminary indication that this is a compressed result.
database schema
{ddl}

question
{question}

external knowledge
{external knowledge}

gold sql (the final resulting SQL)
{gold sql}

cot (content to be compressed)
{cot}

Figure 6: Prompt for compressing CoT.

- **Sample filtering**: We retain only examples that have at least 10 reasoning-enabled outputs with correct SQL, in order to ensure the stability of the statistics.
- **Token counting**: For all reasoning-enabled outputs, we count the number of reasoning tokens, considering only the token consumption of the chain-of-thought within the `<think></think>` tags, and obtain a list R.
- **Lower bound**: We take the 10th percentile of R as the lower bound, i.e.:

$$t_{lower} = Q_{0.1}(R) \tag{8}$$

- **Upper bound**: We take an adaptive quantile of R as the upper bound, i.e.:

$$t_{upper} = Q_p(R) \tag{9}$$

where

$$p = \min(1.0, 0.5 + (1 - reasoning_{ex})). \tag{10}$$

Here, $reasoning_{ex}$ denotes the reliability level of the reasoning outputs: a higher value makes the upper bound closer to the median (yielding a more compressed budget interval), while a lower value leads to a more conservative interval.

After the above procedure, we obtain token budget labels for each example, which are then used to train the token budget model. The distributions of the lower and upper token bounds are shown in Figure 7.

C.2 Feature Definitions

In Section 3.4.2, we model Text-to-SQL task characteristics along three dimensions—database schema, question, and SQL solution space—and obtain 17 features in total. Each feature is detailed as follows:

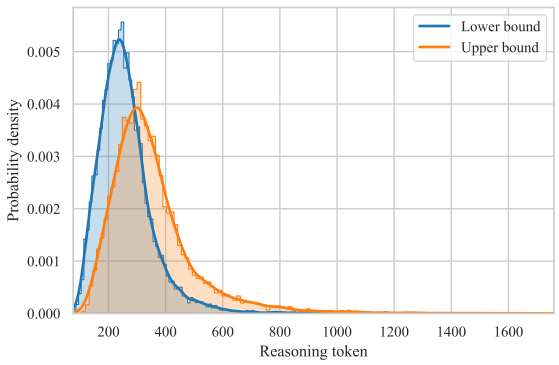


Figure 7: Probability density of lower and upper reasoning token bounds.

Schema features. Let T denote the number of tables in the database, C the total number of columns, and F the number of foreign key relations. We define the following structural complexity features:

- **`table num`**: the number of tables, i.e., T.
- **`column num`**: the number of columns, i.e., C.
- **`avg column num`**: the average number of columns per table, defined as C/T.
- **`foreign key num`**: the number of foreign keys, i.e., F.
- **`foreign key density`**: the foreign key density, defined as F/T, which characterizes the tightness of cross-table connections in the schema.

Question features. For each natural language question q, we extract features from four aspects: length, named entities, numbers, and syntactic structure:

- **`token len`**: the token length of the question.
- **`ner num`**: the number of named entities.
- **`number num`**: the number of numerical expressions.
- **`dependency tree depth`**: the depth of the dependency parse tree. We perform dependency parsing on q, recursively compute the depth of the dependency tree for each sentence root, and take the maximum depth across all sentences as a measure of the question's syntactic complexity.

SQL features. For each query, we execute all sampled SQL outputs and collect those that are semantically correct into a set of correct SQLs, denoted by $\mathcal{S}$. Based on $\mathcal{S}$, we extract the following SQL-level features:

- **`avg token len`**: the average token length of correct SQLs.
- **`avg keyword num`**: the average number of SQL keywords in correct SQLs.
- **`avg join num`**: the average number of `JOIN` operations in correct SQLs.
- **`avg where condition num`**: the average number of `WHERE` conditions in correct SQLs.
- **`avg use table num`**: the average number of tables used in correct SQLs.
- **`avg use column num`**: the average number of columns used in correct SQLs.
- **`avg use table ratio`**: the average table usage ratio of correct SQLs. For each $s \in \mathcal{S}$, we compute the ratio between the number of tables accessed by s and the total number of tables T, and then average this ratio over $\mathcal{S}$.
- **`avg use column ratio`**: the average column usage ratio of correct SQLs. For each $s \in \mathcal{S}$, we compute the ratio between the number of columns accessed by s and the total number of columns C, and then average this ratio over $\mathcal{S}$.

Together, these 17 features characterize the structural complexity of the database, the complexity of question formulation, and the structural complexity of the SQL solution space for each query.

C.3 Correlation Metrics

To systematically analyze the relationship between each feature and reasoning token consumption, we compute three correlation measures for each feature: the Pearson correlation coefficient, the Spearman rank correlation coefficient, and mutual information (MI), with respect to both the lower token bound and the upper token bound.

Pearson correlation. Given a feature vector $\{x_i\}_{i=1}^n$ and a target variable $\{y_i\}_{i=1}^n$, the Pearson correlation coefficient is defined as:

$$\rho_{\text{Pearson}}(x, y) = \frac{\text{cov}(x, y)}{\sigma_x \sigma_y} \tag{11}$$

This metric characterizes the degree of linear correlation between the feature and the target.

Spearman rank correlation. The Spearman rank correlation coefficient first maps x and y to their ranks in the sample, yielding rank sequences $\{r_i^x\}$ and $\{r_i^y\}$, and then computes the Pearson correlation coefficient between them:

$$\rho_{\text{Spearman}}(x, y) = \rho_{\text{Pearson}}(r^x, r^y) \quad (12)$$

This metric is more sensitive to monotonic but nonlinear relationships and measures the consistency of the ordering between the feature and the target without relying on their absolute numerical scales.

Mutual information. Mutual information is used to characterize the degree of statistical dependence between a feature and the target. For the discrete case, it is defined as:

$$I(X;Y) = \sum_x \sum_y p(x,y) \log \frac{p(x,y)}{p(x)\,p(y)} \quad (13)$$

In the continuous case, the summation is replaced by the corresponding integral form. In our implementation, we use `mutual_info_regression` from `scikit-learn` (Pedregosa et al., 2011) to estimate the mutual information between each feature and token budget.

C.4 Correlation Statistics and Feature Selection

Using the three correlation measures described above, we analyze the correlations between each of the 17 features and the lower and upper bounds of the reasoning tokens, respectively. The statistics are summarized in Table 14. We observe that the features most strongly correlated with reasoning length are primarily the question length and those related to SQL structural complexity (e.g., average SQL token length, number of keywords, number of JOINs, and the number and ratio of used columns/tables). These features exhibit consistently high correlations across all three measures.

Consequently, from the 17 features, we select 8 that show the strongest and most consistent correlations with reasoning length for both the lower and upper bounds, and use them to train the token budget prediction model. Examples of their scatter plots with respect to the lower and upper reasoning token bounds are shown in Figure 8.

The data presented in Table 14 and Figure 7 provide empirical support for modeling the lower and upper bounds.

D Experimental Details and Additional Evaluations

D.1 Training Setup

In the cold start stage, we perform supervised fine-tuning with LoRA on SynSQL-ColdStart-16K for 1 epoch, using a learning rate of 5e-6. The LoRA configuration is set to rank=8, α=16, and dropout=0.1. In the reinforcement learning stage, we adopt the TRL framework to conduct GRPO training, sampling 16 outputs per prompt. The learning rate is set to 2e-6 with cosine learning-rate scheduling, and the GRPO hyperparameters are configured as β=0.001 and ϵ=0.2. The LoRA configuration is set to rank=8, α=16, and dropout=0.05.

During the first stage of reinforcement learning, the exploration reward is activated only in the early phase of training (the first 10% of training steps) to avoid interfering with the primary objective of performance optimization.

D.2 Environment

To ensure fair comparison in terms of both performance and efficiency, all experiments are conducted under the same hardware and inference configurations. The server is equipped with an AMD EPYC 9K84 96-Core Processor CPU and 150GB of system memory. Model inference is performed on a single GPU with 96GB of memory. We set the maximum sequence length to 8,192 tokens, with a maximum generation length of 2,048 tokens.

For decoding strategies, we evaluate both Greedy and Self-Consistency. For Self-Consistency, the number of candidates is fixed to 8 and the temperature is set to 0.8, following the original configurations of the baselines.

For the inference framework, we use vLLM (Kwon et al., 2023) 0.6.3 for generation, which matches the original setup of the baselines. Because the Qwen3 series is incompatible with vLLM 0.6.x, we use vLLM 0.8.4 instead for Qwen3-8B, Qwen3-1.7B, Qwen3-4B, as well as Ada-SQL-1.7B and Ada-SQL-4B trained on Qwen3. To avoid potential discrepancies introduced by different runtime versions, the average latency and output token statistics of the baselines reported in Table 3 are computed only over OmniSQL and SQL-R1, while the results of Qwen3-8B are reported for reference only.

Feature	Lower bound			Upper bound		
	Pearson	Spearman	MI	Pearson	Spearman	MI
sql avg token len	0.940	0.936	1.140	0.879	0.913	1.044
sql avg keyword num	0.855	0.862	0.743	0.815	0.866	0.757
sql avg use column num	0.784	0.799	0.555	0.688	0.722	0.448
sql avg join num	0.760	0.773	0.506	0.680	0.725	0.452
sql avg use table num	0.688	0.717	0.406	0.597	0.662	0.364
question token len	0.568	0.534	0.184	0.451	0.438	0.132
sql avg use column ratio	0.563	0.634	0.281	0.498	0.581	0.234
sql avg use table ratio	0.445	0.551	0.259	0.396	0.521	0.220
sql avg where condition num	0.386	0.411	0.146	0.292	0.343	0.140
question dependency tree depth	0.348	0.277	0.066	0.299	0.229	0.034
question ner num	0.243	0.276	0.058	0.148	0.202	0.043
question number num	0.188	0.227	0.044	0.086	0.150	0.032
schema column num	0.061	0.055	0.007	0.043	0.043	0.011
schema table num	0.060	0.052	0.005	0.038	0.038	0.021
schema fk num	0.047	0.045	0.009	0.041	0.041	0.001
schema fk density	0.020	0.020	0.004	0.029	0.026	0.004
schema avg column num	-0.017	-0.013	0.014	-0.005	-0.004	0.000

Table 14: Correlation between Text-to-SQL features and lower/upper bounds of reasoning token usage.

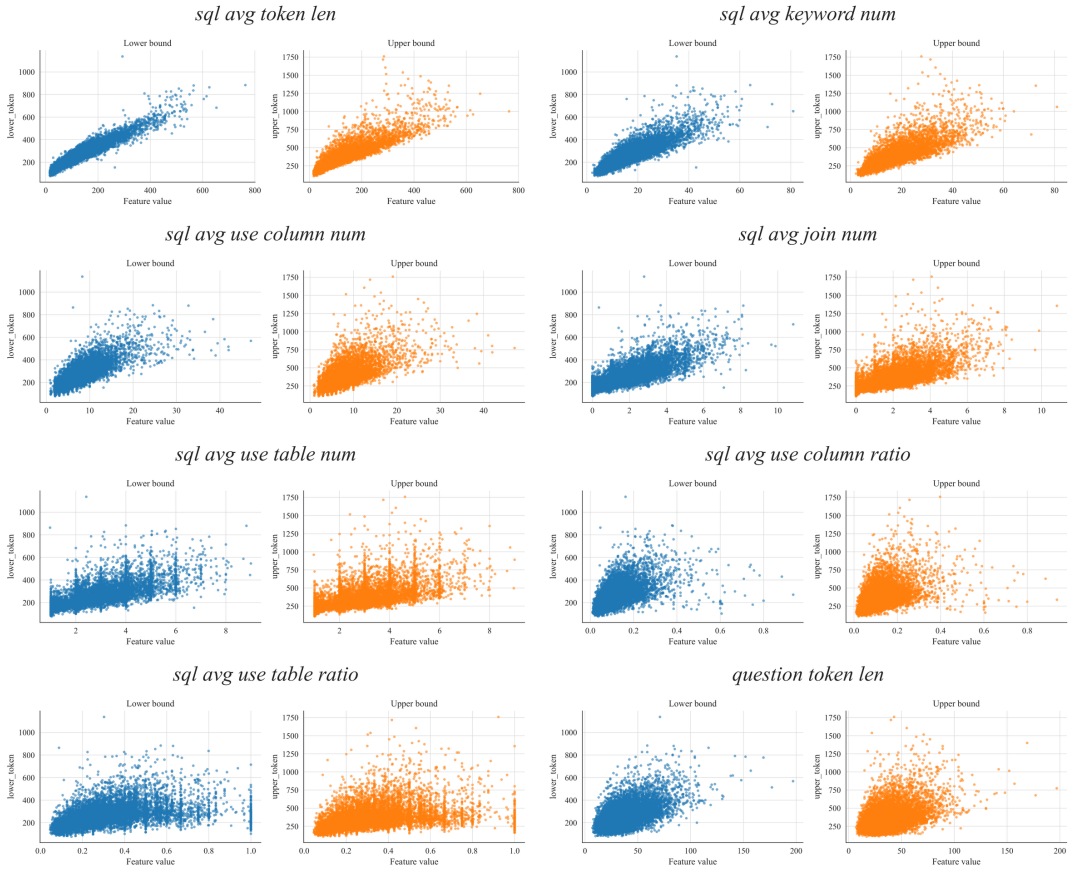


Figure 8: Scatter plots of 8 selected Text-to-SQL features vs. lower/upper reasoning token bounds.

Prompt for Generation SQL

You are a helpful SQL expert assistant. The assistant first thinks about how to write the SQL query by analyzing the question, database schema and external knowledge, then provides the final SQL query.

The reasoning process and SQL query are enclosed within <think> </think> and <answer> </answer> tags respectively. The answer must contain the SQL query within "‘sql...‘" tags.

Database Schema: {ddl}

External Knowledge: {knowledge}

Question: {question}

Figure 9: Prompt for Generation SQL.

D.3 Prompt for SQL Generation

For prompt configuration, OmniSQL and SQL-R1 directly use the prompts provided in their official code repositories. The prompts used by Qwen3 and our Ada-SQL are shown in Figure 9, where *knowledge* denotes the input field provided by BIRD-dev.

D.4 Additional Evaluation

D.4.1 Generalization to the Spider Dataset

To more comprehensively evaluate the generalization capability of Ada-SQL, we conduct additional experiments on the Spider dataset and compare against four representative methods, including Step-by-Step reasoning models STaR-SQL (He et al., 2025) and OmniSQL (Li et al., 2025c), the search-based Alpha-SQL (Li et al., 2025b), and Ellie-SQL (Zhu et al., 2025), the first efficient Text-to-SQL method based on workflow routing. The results are shown in Table 15 and Table 16.

Without training on Spider data, Ada-SQL still demonstrates strong cross-domain generalization ability. Compared with OmniSQL, Ada-SQL achieves comparable execution accuracy while reducing output token consumption by about 70% and inference latency by about 36%. Under the Self-Consistency setting, Ada-SQL still achieves around 62% inference speedup and approximately 70% token reduction compared to OmniSQL. Compared with Ellie-SQL, Ada-SQL further reduces token consumption by about 21% while maintaining higher execution accuracy. These results indicate that Ada-SQL can effectively control inference cost under cross-dataset settings, demonstrating strong generalization of its adaptive reasoning behavior.

Furthermore, we analyze the reasoning mode distribution of Ada-SQL across different query difficulty levels on the Spider dataset, as shown in Table 17. As query complexity increases, the model gradually reduces the use of no-reasoning mode and increasingly adopts short or even long reasoning modes. For Easy queries, approximately 79% of the samples require no explicit reasoning, whereas for Hard and Extra queries, the model almost always activates reasoning. These results demonstrate that Ada-SQL can dynamically adjust reasoning depth according to query complexity, rather than uniformly generating fixed-length reasoning chains, further validating the cross-dataset generalization ability of its adaptive reasoning strategy.

Difficulty	No (%)	Short (%)	Long (%)	EX (%)
Easy	79.44	20.56	0	93.15
Medium	32.74	67.26	0	88.34
Hard	5.17	94.83	0	74.71
Extra Hard	0	98.19	1.81	61.45

Table 17: Reasoning mode distribution and execution accuracy across different Spider query difficulty levels (No: no reasoning, Short: short reasoning, Long: long reasoning).

D.4.2 Evaluation on BIRD Mini-Dev

We compare Ada-SQL with OmniSQL on BIRD Mini-Dev, a high-quality subset of BIRD. As shown in Table 18, Ada-SQL achieves 63.0% EX, while reducing average token consumption from 692.05 to 278.37 (a 59.78% reduction).

D.5 Budget Adherence Analysis

We further analyze Ada-SQL's adherence to token budget constraints during training and inference. As shown in Table 19, the budget adherence ratio steadily increases from 71.67% to 93.23% during training and gradually stabilizes. Although no explicit budget signal is provided during inference, 90.49% of the generated samples still satisfy the budget constraint. This indicates that the token

Method	Size	Performance	Efficiency	
		EX (%)	Avg Latency (s)	Avg Output Tokens
Greedy				
STaR-SQL	8B	75.0	-	-
OmniSQL	7B	**85.88**	0.91	535.20
Ada-SQL	7B	82.88	**0.58** (↓ 36.26%)	**161.92** (↓ 69.75%)
Self-Consistency				
STaR-SQL @16	8B	86.6	-	-
Alpha-SQL SC	7B	84.0	-	-
OmniSQL @8	7B	**86.85**	4.00	4287.38
Ada-SQL @8	7B	86.17	**1.52** (↓ 62.00%)	**1279.91** (↓ 70.15%)

Table 15: Comparison with STaR-SQL, Alpha-SQL, and OmniSQL on the Spider-dev. Latency and token consumption are reported under greedy and self-consistency (SC) decoding settings.

Method	EX(%)	Avg. WT
EllieSQL	76.31	1906.79
Ada-SQL(Gre)	**82.88**	**1503.43**(↓ 21.15%)

Table 16: Comparison with the routing-based efficient Text-to-SQL system EllieSQL on Spider-dev. EllieSQL results are taken from the original paper.

Method	EX(%)	Avg. Output Tokens
OmniSQL	61.8	692.05
Ada-SQL	**63.0**	**278.37**(↓ 59.78%)

Table 18: Comparison between Ada-SQL and OmniSQL on BIRD Mini-Dev.

budget signal has been effectively internalized as a generation preference of the model.

Stage	Step	Budget Adherence (%)
Training	0–10K	71.67
	10–20K	87.13
	20–30K	90.16
	30–40K	91.18
	>40K	93.23
Inference		90.49

Table 19: Token budget adherence ratios during training and inference.

E Case Study

E.1 Per-example Analysis of Reasoning Token Usage

We collect statistics on the reasoning token consumption per example on BIRD-dev for SQL-R1, OmniSQL, and Ada-SQL, as shown in Figure 10. We observe that, compared with SQL-R1 and OmniSQL, Ada-SQL uses fewer reasoning tokens on most examples, leading to a more controlled overall reasoning cost.

E.2 No Reasoning Case

For structurally simple queries that do not require multi-step reasoning, Ada-SQL can directly generate the correct SQL without producing intermediate reasoning steps, thereby avoiding unnecessary token consumption. An example is shown in Figure 11.

E.3 Short Reasoning Case

For moderately complex queries, Ada-SQL generates concise reasoning steps to guide SQL construction, controlling the reasoning length while maintaining correctness. An example is shown in Figure 12.

E.4 Long Reasoning Case

For complex queries that require multi-step reasoning, Ada-SQL allocates a longer reasoning chain to progressively construct the SQL, thereby improving the correctness of the generated results. An example is shown in Figure 13.

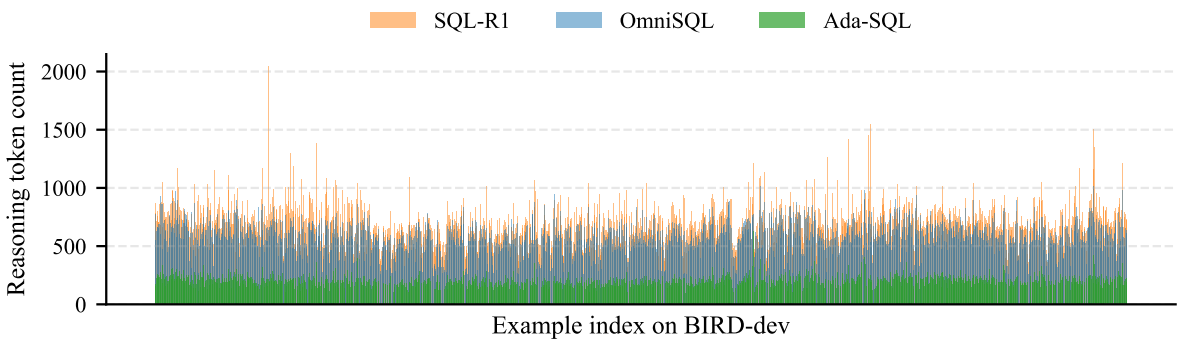


Figure 10: Reasoning token usage for Ada-SQL, SQL-R1, and OmniSQL on each example of BIRD-dev.

Case 1: No Reasoning

Difficulty: Simple

Question: List the account IDs with monthly issuance of statements.

Model Output

```
<think>
</think>
<answer>
SELECT account_id
FROM account
WHERE frequency = 'POPLATEK MESICNE';
</answer>
```

Reasoning Tokens: 0

Figure 11: Example where Ada-SQL skips reasoning for a simple query.

Case 2: Short Reasoning

Difficulty: Moderate

Question: Of all the cards that are designed by Aaron Miller, how many of them are incredibly powerful?

Model Output

````
<think>
1. The question asks for cards designed by Aaron Miller, so we filter rows
where `artist = 'Aaron Miller'`.
2. Cards are considered "incredibly powerful" if both `cardkingdomfoilid`
and `cardkingdomid` are not null.
3. To count the number of such cards, we use the `COUNT(*)` function.

Here is the SQL query:
```sql
SELECT COUNT(*)
FROM cards
WHERE artist = 'Aaron Miller'
  AND cardkingdomfoilid IS NOT NULL
  AND cardkingdomid IS NOT NULL;
```
</think>
<answer>
```sql
SELECT COUNT(*)
FROM cards
WHERE artist = 'Aaron Miller'
  AND cardkingdomfoilid IS NOT NULL
  AND cardkingdomid IS NOT NULL;
```
</answer>
````

Reasoning Tokens: 121

Figure 12: Example where Ada-SQL use short reasoning for a moderate query.

Case 3: Long Reasoning

Difficulty: Challenging

Question: For the customer who paid 634.8 in 2012/8/25, what was the consumption decrease rate from Year 2012 to 2013?

Model Output

<think>
1. The query involves calculating the consumption decrease rate for a specific customer.
2. The customer is identified by the payment amount of 634.8 on '2012-08-25' in the `transactions_1k` table.
3. The `yearmonth` table contains the `consumption` values for 2012 and 2013, which are needed for the calculation.
4. Join the `transactions_1k` table with the `yearmonth` table on `customerid` to link the payment to the consumption data.
5. Filter the `transactions_1k` table for the specific date '2012-08-25' and amount 634.8.
6. Use conditional aggregation to calculate the consumption for 2012 and 2013, then compute the decrease rate.

Here's how the query can be written:

```sql
SELECT
  (SUM(CASE WHEN T2.date LIKE '2012%' THEN T2.consumption ELSE 0 END) -
   SUM(CASE WHEN T2.date LIKE '2013%' THEN T2.consumption ELSE 0 END)) /
  SUM(CASE WHEN T2.date LIKE '2012%' THEN T2.consumption ELSE 0 END)
  AS consumption_decrease_rate
FROM transactions_1k AS T1
INNER JOIN yearmonth AS T2
ON T1.customerid = T2.customerid
WHERE T1.date = '2012-08-25' AND T1.price = 634.8;
```

</think>
<answer>

```sql
SELECT
  (SUM(CASE WHEN T2.date LIKE '2012%' THEN T2.consumption ELSE 0 END) -
   SUM(CASE WHEN T2.date LIKE '2013%' THEN T2.consumption ELSE 0 END)) /
  SUM(CASE WHEN T2.date LIKE '2012%' THEN T2.consumption ELSE 0 END)
  AS consumption_decrease_rate
FROM transactions_1k AS T1
INNER JOIN yearmonth AS T2
ON T1.customerid = T2.customerid
WHERE T1.date = '2012-08-25' AND T1.price = 634.8;
```

</answer>

Reasoning Tokens: 355

Figure 13: Example where Ada-SQL use long reasoning for a challenging query.