Learning from Imperfect Data: Towards Efficient Knowledge Distillation of Autoregressive Language Models for Text-to-SQL

Anonymous ACL submission

Abstract

Large Language Models (LLMs) have shown promising performance in text-to-SQL, which involves translating natural language questions into SQL queries. However, current text-to-SQL LLMs are computationally expensive and challenging to deploy in real-world applications, highlighting the importance of compressing them. To achieve this goal, knowledge distillation (KD) is a common approach, which aims to distill the larger teacher model into a smaller student model. While numerous KD methods for autoregressive LLMs have emerged recently, it is still under-explored whether they work well in complex text-to-SQL scenarios. To this end, we conduct a series of analyses and reveal that these KD methods generally fall short in balancing performance and efficiency. In response to this problem, we propose to improve the **KD** with **I**mperfect **D**ata, namely `KID`, which effectively boosts the performance without introducing much training budget. The core of `KID` is to efficiently mitigate the training-inference mismatch by simulating the cascading effect [1] of inference in the imperfect training data. Extensive experiments on 5 text-to-SQL benchmarks show that, `KID` can not only achieve consistent and significant performance gains (up to +5.83% average score) across all model types and sizes, but also effectively improve the training efficiency.

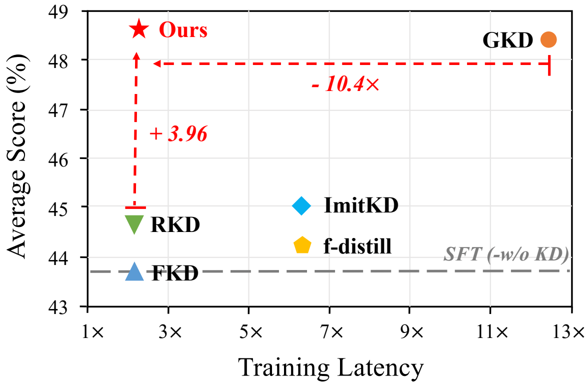

Figure 1: **Comparisons of different KD methods** for distilling the student model (QWen1.5-0.5B) from the teacher (QWen1.5-4B). The x-axis denotes the training latency relative to the SFT baseline, while the y-axis denotes the average performance of students on several popular text-to-SQL benchmarks. The evaluation details are in §4. We see that our method achieves the best trade-off between performance and efficiency.

1 Introduction

Text-to-SQL, which aims to translate a user's natural language question into an executable and accurate SQL query, is a transformative application of large language models (LLMs) (Katsogiannis-Meimarakis and Koutrika, 2023; Li et al., 2024a; Pourreza and Rafiei, 2024). However, with the scaling of model size, the inference and deployment of LLM-based text-to-SQL systems become more computationally expensive and memory intensive, hindering the development of real-world industrial applications that require low inference latency (Sun et al., 2023b). Hence, it is crucial and green to compress these text-to-SQL LLMs and accelerate the inference, while not losing much performance (Schwartz et al., 2020; Zhu et al., 2023).

A common model compression approach is knowledge distillation (KD), which involves compressing a large teacher model by distilling its knowledge into a small student model (Hinton et al., 2015; Kim and Rush, 2016). Recently, numerous KD methods for autoregressive LLMs have emerged (Gu et al., 2023; Agarwal et al., 2024; Xu et al., 2024), but most of them focus on the general instruction-tuning scenarios. Different from the general tasks that allow for flexible and diverse outputs, text-to-SQL is more challenging, as it requires the LLMs to precisely output the table/column name. Even a minor error in the SQL query could lead to the wrong result. Unfortunately,

[1]The error at the early step will affect the future predictions during the autoregressive inference (Agarwal et al., 2024).

it is still under-explored whether these KD methods work well for text-to-SQL LLMs.

To this end, we conduct preliminary experiments by applying 5 representative KD methods to distill the QWen-family LLMs (Bai et al., 2023) on the popular text-to-SQL benchmark, *i.e.*, Spider (Yu et al., 2018). We find that the performance gains of these KD methods mainly rely on the model-generated data, which is effective but hard to obtain. Specifically, although the model-generated data can alleviate the training-inference mismatch (*i.e.*, difference between teacher-forcing training and autoregressive inference (Pang and He, 2020)) and achieves remarkable performance, it requires the student model to autoregressively generate in an online fashion, leading to unbearable training latency. As illustrated in Figure 1, GKD (Agarwal et al., 2024) training with model-generated data performs well but greatly suffers from training inefficiency. Thus, there raises a question: *whether we can mitigate the training-inference mismatch more efficiently*?

Motivated by this, we propose a simple-yet-effective approach to improve KD, namely `KID`, and achieve a better trade-off between performance and efficiency. The core of `KID` is to force the student to rewrite the ground-truth training data into imperfect one, and then learn how to calibrate these imperfect data. Intuitively, by introducing some errors in the imperfect data, we can simulate the cascading effect of inference during training processes, thus mitigating the training-inference mismatch. More specifically, instead of autoregressively generating the on-policy data, the generation processes of imperfect data only require one-pass forward, which is more efficient and affordable. Moreover, by doing so, we can also encourage the student to learn how to calibrate these imperfect tokens and further improve the KD performance.

We evaluate `KID` on a variety of popular text-to-SQL benchmarks, including BIRD (Li et al., 2024b), Spider (Yu et al., 2018) and its variants, upon 3 types of autoregressive LLMs: QWen (Bai et al., 2023), CodeGen (Nijkamp et al., 2022) and LLaMA (Touvron et al., 2023). Results show that `KID` can not only achieve a better trade-off between performance and efficiency, but also bring consistent and significant improvements (up to +5.83% average score) among all model types and sizes. Moreover, compared to the standard KD, `KID` can effectively improve the robustness of students.

Contributions. Our main contributions are:

- We reveal that current KD methods for text-to-SQL LLMs generally fall short in balancing performance and efficiency.
- We propose a simple-yet-effective approach (`KID`) to effectively improve KD performance without introducing much training budget.
- Extensive experiments show that `KID` outperforms the standard KD by a large margin and effectively improves the student's robustness.

2 Preliminary

2.1 Task Formulation

Text-to-SQL aims to convert a natural language question $\mathcal{Q}$ into a SQL query $\mathcal{Y}$, which is executable and can accurately retrieve relevant data from a database $\mathcal{D}$. The database $\mathcal{D}$ usually contains the schema (*i.e.*, tables and columns) and metadata, containing column types/values, primary keys, foreign key relations and *etc* (Zhong et al., 2017). Specifically, given an LLM $\mathcal{M}$ and a prompt template $\mathcal{P}$, we enforce the $\mathcal{M}$ to autoregressively generate an output sequence $\mathcal{Y}$ conditioned on the $\mathcal{P}(\mathcal{Q}, \mathcal{D})$, which can be formulated as:

$$\mathcal{Y}_t \sim \mathbb{P}_{\mathcal{M}}(\mathcal{Y}_t \mid \mathcal{P}(\mathcal{Q}, \mathcal{D}), \mathcal{Y}_{<t}), \tag{1}$$

where $\mathbb{P}_{\mathcal{M}}(\mathcal{Y}_t \mid \mathcal{P}(\mathcal{Q}, \mathcal{D}), \mathcal{Y}_{<t})$ is the probability for the next token, and $\mathcal{Y}_t$ is the t-th token of $\mathcal{Y}$.

2.2 Knowledge Distillation of LLMs

Knowledge Distillation (KD) aims to compress a large teacher model $\mathcal{M}_p$ by distilling its knowledge into a small student model $\mathcal{M}_q^\theta$ parameterized by θ. Given a divergence function $\mathcal{F}$ and a training set $\mathcal{G}$, we can train the student model as follows:

$$\theta^* := \arg\min \mathbb{E}_{(x,y)\sim\mathcal{G}}[\mathcal{F}(\mathcal{M}_q \| \mathcal{M}_q^\theta)(y|x)], \tag{2}$$

where (x, y) is the task-specific input-output pair[2] of $\mathcal{G}$, and $\mathcal{F}(\mathcal{M}_q \| \mathcal{M}_q^\theta)(y|x) = \frac{1}{|y|}\sum_{t=1}^{|y|} \mathcal{F}\big(p(\cdot \mid x, y_{<t}) \| q^\theta(\cdot \mid x, y_{<t})\big)$ is the divergence between the teacher and student distributions, denoted as p and q^θ, respectively. The choices of training set $\mathcal{G}$ and divergence function $\mathcal{F}$ give rise to different possible KD algorithms, *e.g.*, Forward KD (FKD) (Hinton et al., 2015), Reverse KD (RKD) (Gu et al., 2023),

[2]For text-to-SQL task in §2.1, x refers to the input question $\mathcal{P}(\mathcal{Q}, \mathcal{D})$ and y refers to the output SQL query $\mathcal{Y}$.

Method	**Divergence**	**Training Dataset**
Data type: Fixed dataset		
FKD	FKL	Ground-truth data
RKD	RKL	Ground-truth data
Data type: Model-generated dataset		
f-distill	TVD	Data generated by $\mathcal{M}_p$ and $\mathcal{M}_q^\theta$
ImitKD	FKL	Ground-truth+data generated by $\mathcal{M}_q^\theta$
GKD	FKL/RKL/JSD	On-policy data generated by $\mathcal{M}_q^\theta$
KID	RKL	Imperfect ground-truth data

Table 1: **Summary of various KD algorithms in terms of training data and divergence**. Notably, $\mathcal{M}_p$ and $\mathcal{M}_q^\theta$ denote the teacher and student models, respectively.

f-distill (Wen et al., 2023), ImitKD (Lin et al., 2020) and GKD (Agarwal et al., 2024). The summary of these representative KD algorithms is shown in Table 1.

The common divergences for KD contain the Forward Kullback-Leibler (FKL) (Van Erven and Harremos, 2014), Reverse KL (RKL) (Malinin and Gales, 2019), Jensen–Shannon divergence (JSD) (Fuglede and Topsoe, 2004) and total variation distance (TVD) (Verdú, 2014). The details of these divergences can be found in Appendix A.3. On the other hand, $\mathcal{G}$ may consist of input-output pairs in the original training set (denoted as **ground-truth dataset**), or sequences generated from teacher $\mathcal{M}_p$ or student $\mathcal{M}_q^\theta$ (denoted as **model-generated dataset**). For the data generated by $\mathcal{M}_p$, we feed the input into the $\mathcal{M}_p$ and obtain the teacher's output beforehand and keep them fixed during training. Conversely, for the data generated by $\mathcal{M}_q^\theta$, since the student is continuously updated, we obtain the student's output in an online fashion. Such online generated data is also called "on-policy data" by Agarwal et al. (2024).

2.3 Empirical Analyses

As mentioned in §1, it is under-explored whether the aforementioned KD algorithms work well for text-to-SQL LLMs. Hence, we conduct preliminary experiments to investigate it in this part.

Setting. We conduct experiments by first fine-tuning larger LLMs on the original training dataset as teachers. Then, we use different KD methods to distill a smaller student with the teacher's guidance. Here, we use the QWen1.5-0.5B (Bai et al., 2023) as the student and use the other QWen-family models (*i.e.*, QWen1.5-1.8B/-4B/-7B) as teachers. Spider (Yu et al., 2018) is used as training data, and the models are evaluated on the development set.

Method	Divergence	1.8B	4B	7B
Training data: ***Fixed dataset***				
FKD	FKL	57.3	57.4	57.3
RKD	RKL	62.7	60.1	61.5
Training data: ***Model-generated dataset***				
f-distill	TVD	57.6	58.6	59.6
ImitKD	FKL	58.3	59.5	59.1
GKD-FKL	FKL	61.1	62.1	60.7
GKD-RKL	RKL	**62.9**	**63.8**	**64.3**
GKD-JSD	JSD	62.8	62.7	**64.3**

Table 2: **Preliminary experimental results (%) of various KD methods**. We report the execution accuracy of QWen1.5-0.5B distilling from QWen1.5-{1.8B, 4B, 7B} on the Spider benchmark. Best results are in **bold**.

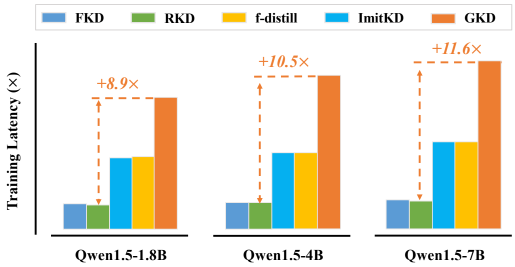

Figure 2: **Comparisons of training latency between various KD methods**. The x-axis denotes the teacher models, and the y-axis denotes the training latency relative to the SFT baseline. For ease of illustration, we only report the results of RKL divergence for GKD.

We follow (Li et al., 2024a) and use the "Execution Accuracy" as metric to quantify the model output.

Findings. The contrastive results are listed in Table 2, from which we empirically find that:

Reverse KL is more suitable for distilling the text-to-SQL LLMs. We first analyze the impact of different divergence functions, and find that RKL generally outperforms the other divergences, *e.g.*, FKD (57.4%) *v.s.* RKD (60.1%) and GKD-FKL (62.1%) *v.s.* GKD-RKL (63.8%). This is similar to the statements of prior studies (Gu et al., 2023; Wu et al., 2024), as they argue that Reverse KL shows mode-seeking behaviors, *i.e.*, it does not force the student to fit all teacher's distributions, but assigns high probabilities to teacher's large modes and ignores the small ones. In the context of text-to-SQL, the output tokens (*e.g.*, table/column name and value) are usually precise and low-diversity, and enforcing the student to learn the high-probability regions could lead to better performance.

Model-generated datasets perform better but suffer from training inefficiency. By compar-

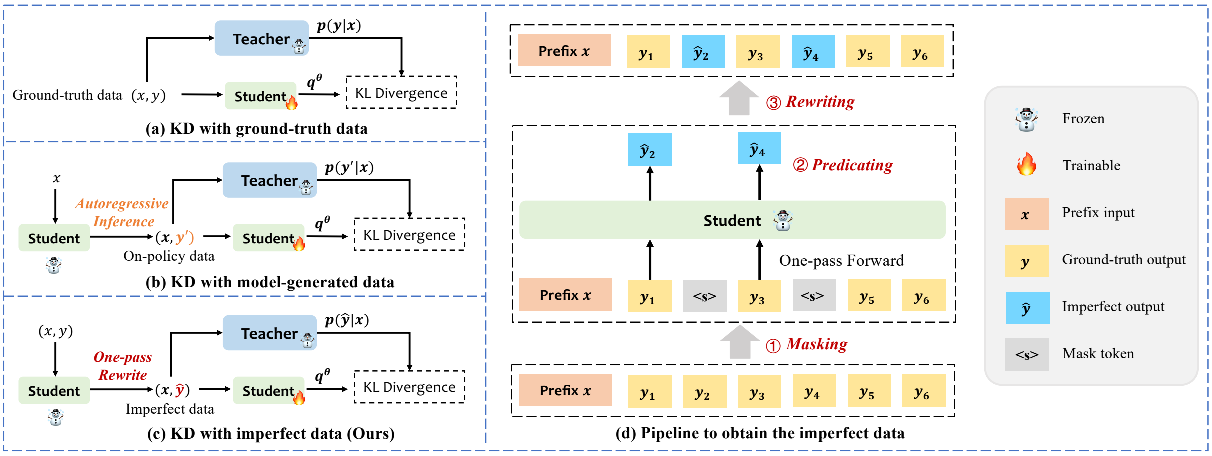

Figure 3: **Illustrations of different KD methods**: **(a)** KD methods with ground-truth data, **(b)** KD methods with model-generated data and **(c)** our KID method with imperfect data. Additionally, we show **(d)** the pipeline to obtain the imperfect data, which contains three-stage processes: ❶ ***masking***, ❷ ***predicting*** and ❸ ***rewriting***.

ing the KD results between ground-truth datasets and model-generated datasets, we find that model-generated datasets perform better than the fixed ground-truth ones, especially the on-policy dataset generated by students (*i.e.*, GKD). This is because that student-generated dataset can alleviate the training-inference mismatch, *i.e.*, the discrepancy between teacher-forcing training and free-run inference. Despite its remarkable performance, it requires the student to autoregressively generate the output in an online manner, which will lead to unaffordable training latency. This can be empirically proven by the results in Figure 2, as the training latency of GKD is much higher than those trained on ground-truth datasets.

3 Improving Knowledge Distillation with Imperfect Data

Motivation and Overview. Based on the observation in §2, we recognize that the key for improving the performance KD is to alleviate the training-inference mismatch. However, the current KD methods relying on model-generated datasets usually suffer from training inefficiency, *i.e.*, they fail to balance the performance and efficiency. Thus, there raises a question: *whether we can mitigate the training-inference mismatch more efficiently*? Motivated by this, we propose to improve KD with imperfect data (KID), which effectively and efficiently boosts the performance by simulating the cascading effect of inference during training. The illustration of KID is shown in Figure 3.

Intuition of KID. As stated by prior studies (Pang and He, 2020; Agarwal et al., 2024), the training-inference mismatch mainly comes from the cascading effect of inference. Specifically, during training, LLMs condition on ground-truth tokens. However, during inference, they condition on the model-generated tokens, which might be wrong and affect the future predictions. Intuitively, enforcing the student to rewrite the ground-truth training data into imperfect one, *i.e.*, introducing some errors during training, can simulate the cascading effect of inference during and thus mitigate the training-inference mismatch. Moreover, by encouraging the student to learn how to calibrate these imperfect tokens, KID can further improve the performance.

Pipeline to Obtain the Imperfect Data. The key technique of KID is to rewrite the ground-truth data into an imperfect one. Specifically, the generation of imperfect data consists of three-stage processes: ❶ ***masking***, ❷ ***predicting*** and ❸ ***rewriting***. In practice, we ❶ first sample α of tokens[3] from the ground-truth output y and mask them with a special token (*e.g.*, "<s>"). For sampling the tokens, we design some strategies: 1) "Random": randomly sampling, 2) "Uniform": uniformly sampling, 3) "Hard": sampling α of tokens with the lowest confidence; 4) "Easy": sampling α of tokens with the highest confidence. More specifically, for 3) and 4), we feed the original sequence y into the student for obtaining prediction probabilities q_i^θ, and then compute the entropy of q_i^θ as the confidence[4].

After masking the spans of y, we ❷ then gener-

[3]The analysis of sampling ratio α can be found in §4.3.

[4]Intuitively, the tokens with high entropy value are hard-to-learn, as the model predict them with low confidence towards the gold labels (Zhong et al., 2023).

ate imperfect tokens to fill in the spans. Specifically, we feed the masked sequence into the student to generate predictions with a one-pass forward process. Finally, given the predicted imperfect tokens on the masking place, we ❸ rewrite the ground-truth y into the imperfect one $\hat{y}$.

Training of KID. During training, given a mini-batch of input-output pairs (x, y), we first perform the above processes to obtain the imperfect data $(x, \hat{y})$. Then, we can train the student model with the teacher's guidance. As shown in §2, Reverse KL is more suitable for text-to-SQL task, and we thus use it as the divergence function in our KID. Moreover, since our KID require sampling from a student, which may generate poor samples at the beginning of training and make the distilling more difficult, we follow prior works (Wen et al., 2023; Gu et al., 2023) and combine the KD loss in Eq. 2 with an auxiliary maximum likelihood estimation (MLE) loss. Specifically, the MLE loss enforces the student to predict the ground-truth target sequences y. Notably, for a fair comparison, we also add the auxiliary MLE loss into the baseline KD methods that rely on the ground-truth data.

4 Experiments

4.1 Setup

Tasks and Datasets. We conduct our main experiments on two popular text-to-SQL benchmarks, *i.e.*, Spider (Yu et al., 2018) and BIRD (Li et al., 2024b). For each task, models are trained with the original training set and evaluated on the development set, denoted as Spider-dev and BIRD-dev, respectively. Moreover, following prior studies (Li et al., 2023, 2024a), we also evaluate the models trained with the Spider dataset on three more challenging robustness benchmarks, *i.e.*, Spider-DK (Gan et al., 2021b), Spider-Realistic (Deng et al., 2021) and Spider-Syn (Gan et al., 2021a).

For evaluation on Spider-family benchmarks, we utilize two widely-used metrics, *i.e.*, "Execution Accuracy" (EX) (Yu et al., 2018) and "Test-Suite Accuracy" (TS) (Zhong et al., 2020). For BIRD, we simply use the EX as the evaluation metric. Notably, BIRD offers external knowledge for guiding the generation of SQL queries. Considering that such external knowledge is usually unavailable in the real world, we follow Li et al. (2024a) and perform the evaluation in two settings: without ("w/o EK") and with ("w/ EK") external knowledge. The details of all tasks are shown in Appendix A.1.

Models. We evaluate KID on three types of LLMs with various sizes: QWen1.5 (Bai et al., 2023) (*student*: 0.5B, *teachers*: 1.8B, 4B, 7B), CodeGen (Nijkamp et al., 2022) (*student*: 350M, *teachers*: 2B), and LLaMA2 (*student*: TinyLLaMA-1.1B (Zhang et al., 2024b)[5], *teachers*: 7B (Touvron et al., 2023)). All models are trained with a popular parameter-efficient fine-tuning method, *i.e.*, LoRA (Hu et al., 2021). The details of all training hyper-parameters can be found in Appendix A.2.

Baselines. We consider 5 cutting-edge KD baselines in our main experiment: Forward KD (FKD) (Hinton et al., 2015), Reverse KD (RKD) (Gu et al., 2023), f-distill (Wen et al., 2023), ImitKD (Lin et al., 2020) and GKD[6] (Agarwal et al., 2024). For reference, we also report the performance of teachers as the upper bound. We use the codebase of Liu et al. (2023) to implement these baselines and distill students.

4.2 Main Results

KID achieves a better trade-off between the KD performance and efficiency. The main results on QWen-family models are listed in Table 3. As seen, most KD methods outperform the SFT baseline, while introducing extra training budgets. Training with the on-policy data, GKD achieves much better performance than the other counterparts. However, the computational budget of GKD is not affordable, as it leads to up to 13.9× training latency against the SFT baseline. Conversely, our KID can not only achieve comparable or even better performance than GKD, but also effectively reduce the training latency. These results can prove the superiority of our method.

KID brings consistent and significant performance gains among all model sizes and types. In addition to QWen-family models, we also apply our method on CodeGen and LLaMA models, and report the results in Table 4. Notably, due to the space limitation, we only report the contrastive results of two most relevant KD counterparts, *i.e.*, RKD and GKD. From the results of Table 3 and 4, it can be found that our KID consistently outperforms the other KD counterparts and brings significant performance gains (up to +5.83% average score)

[5] Since there are no existing official LLaMA smaller than 7B, we use the other re-produced smaller TinyLLaMA-1.1B from Zhang et al. (2024b) as the student.

[6] As shown in Table 2, GKD with RKL divergence (*i.e.*, GKD-RKL) performs best, and we thus only report the results of GKD-RKL for GKD in the following content.

Method	Latency	Spider-dev		BIRD-dev (EX%)		Spider-DK		Spider-Real		Spider-Syn		Score	
		EX%	TS%	w/o EK	w/ EK	EX%	TS%	EX%	TS%	EX%	TS%	Avg.	Δ
Student: ***QWen1.5-0.5B***													
SFT	1.0×	57.8	56.4	16.36	30.51	44.8	46.5	50.6	47.6	44.2	43.7	43.85	*
Teacher: ***QWen1.5-1.8B***													
Teacher	1.5×	67.3	66.3	21.71	34.22	54.6	52.3	62.0	60.8	52.7	52.6	52.45	-
FKD	2.1×	57.3	56.5	16.82	28.68	43.7	41.7	50.2	48.0	43.7	43.3	42.99	-0.86
RKD	2.0×	62.7	61.5	16.10	31.81	50.8	49.2	51.2	49.6	48.7	48.3	46.99	+3.14
f-distill	6.0×	57.6	56.3	15.78	27.90	45.0	43.2	52.6	51.0	43.4	43.0	43.58	-0.27
ImitKD	5.9×	58.3	57.2	16.04	28.49	46.2	44.1	52.4	50.8	44.1	43.3	44.09	+0.24
GKD	10.9×	62.9	61.6	18.25	32.99	**49.9**	**47.9**	50.6	48.6	**48.6**	**48.1**	46.94	+3.09
KID (Ours)	2.0×	**63.7**	**63.1**	**18.38**	**33.12**	47.6	45.4	**53.0**	**51.4**	47.5	47.0	**47.02**	**+3.17**
Teacher: ***QWen1.5-4B***													
Teacher	3.0×	78.2	77.3	35.27	48.11	61.3	58.7	72.6	70.3	67.4	66.8	63.60	-
FKD	2.2×	57.4	56.5	18.32	29.34	47.1	45.6	50.6	48.6	42.4	41.8	43.77	-0.08
RKD	2.2×	60.1	59.1	17.01	31.75	45.8	43.6	49.6	47.4	46.1	45.6	44.61	+0.76
f-distill	6.3×	58.6	57.3	17.67	31.55	45.8	43.6	50.8	49.2	44.4	43.8	44.27	+0.42
ImitKD	6.3×	59.5	59.4	19.04	30.31	48.6	46.9	49.2	46.9	45.0	44.5	44.94	+1.09
GKD	12.7×	63.8	62.4	**20.21**	**36.11**	**50.8**	**48.2**	**55.5**	**53.3**	47.5	46.9	48.47	+4.62
KID (Ours)	2.3×	**65.8**	**64.7**	20.08	33.57	50.5	48.0	55.1	**53.3**	**47.6**	**47.0**	**48.57**	**+4.72**
Teacher: ***QWen1.5-7B***													
Teacher	3.3×	81.6	80.6	39.44	52.02	67.7	64.9	76.6	74.2	70.1	69.5	67.67	-
FKD	2.4×	57.3	56.4	17.14	31.03	46.4	44.9	50.6	49.0	41.0	40.5	43.43	-0.42
RKD	2.3×	61.5	60.2	16.10	31.81	48.4	46.5	51.0	49.2	46.7	46.0	45.74	+1.89
f-distill	7.2×	59.6	58.2	18.19	32.78	47.7	46.0	49.8	47.6	44.9	44.4	44.92	+1.07
ImitKD	7.2×	59.1	57.9	17.60	30.44	47.3	45.4	48.8	47.2	43.8	43.4	44.09	+0.24
GKD	13.9×	**64.3**	**62.9**	20.08	**34.62**	**51.6**	**49.7**	**54.1**	**51.6**	46.9	46.2	**48.20**	**+4.35**
KID (Ours)	2.3×	64.0	62.6	**20.40**	34.35	50.7	48.5	52.4	50.8	**47.7**	**47.3**	47.88	+4.03

Table 3: **Evaluation of QWen-family models on several popular text-to-SQL benchmarks**. Notably, "Latency" means the average training latency relative to the SFT baseline. "Spider-Real" refers to the Spider-Realistic benchmark. "Avg." denotes the average performance among all benchmarks and "Δ" denotes the performance gains against the SFT baseline. Best performance in each group is emphasized in **bold**.

against the SFT baseline among all model sizes and types, indicating its universality.

KID effectively improves the robustness of distilled models. Spider-DK, Spider-Syn, and Spider-Realistic are widely-used challenging benchmarks to investigate the robustness of text-to-SQL models. Contrastive results on these benchmarks show that our KID exhibits exceptional performance and effectively improves the robustness of distilled students. For example, when distilling CodeGen models, KID achieves gains of 2.7% on Spider-DK (43.7% to 46.4%) and 2.1% on Spider-Realistic (45.5% to 47.6%), comparing with the best counterpart.

4.3 Analysis of KID

We evaluate the impact of each component of our KID, including 1) masking strategies, 2) masking ratio α, and 3) rewriting approach for obtaining the imperfect data. Additionally, we 4) perform the in-depth analysis on the training efficiency of KID.

Figure 4: **Analysis of different masking strategies**. The y-axis denotes the EX performance on Spider-dev. For reference, we also report the results of SFT.

Effect of different masking strategies. As mentioned in §3, we introduce several strategies to select the tokens for masking. Here, we conduct experiments to analyze the impact of different masking strategies. Results of CodeGen-350M and TinyLLaMA-1.1B in Figure 4 show that: 1) Our KID with various masking strategies consis-

Method	Latency	Spider-dev		BIRD-dev (EX%)		Spider-DK		Spider-Real		Spider-Syn		Score	
		EX%	TS%	w/o EK	w/ EK	EX%	TS%	EX%	TS%	EX%	TS%	Avg.	Δ
Student: ***CodeGen-350M***, *Teacher:* ***CodeGen-2B***													.
SFT	1.0×	53.1	51.8	9.90	26.01	37.4	36.1	38.4	36.0	35.4	34.9	35.90	*
Teacher	3.7×	72.3	71.3	26.47	35.66	57.9	55.1	63.2	61.6	55.4	54.8	55.37	-
RKD	2.1×	55.1	54.4	10.50	27.18	43.6	40.0	43.1	40.7	37.6	36.8	38.90	+3.00
GKD	14.1×	56.6	54.9	**11.44**	**27.57**	43.7	40.4	45.5	43.1	40.1	39.3	40.26	+4.36
KID (Ours)	2.4×	**58.4**	**56.8**	10.52	**27.57**	**46.4**	**44.1**	**47.6**	**44.5**	**41.1**	**40.3**	**41.73**	**+5.83**
Student: ***TinyLLaMA-1.1B***, *Teacher:* ***LLaMA2-7B***													.
SFT	1.0×	63.0	61.8	13.40	24.77	49.0	48.0	54.7	52.4	51.4	50.6	46.91	*
Teacher	2.6×	78.8	77.9	35.40	48.63	64.5	61.1	72.4	70.1	67.6	66.4	64.28	-
RKD	1.4×	66.0	64.6	15.45	31.75	48.4	46.9	55.7	54.1	52.9	52.2	48.80	+1.89
GKD	8.3×	64.8	63.2	16.62	**33.44**	52.1	49.9	54.1	51.0	53.0	51.8	49.00	+2.09
KID (Ours)	1.5×	**68.1**	**66.8**	**18.97**	32.53	**52.9**	**51.8**	**59.8**	**57.7**	**55.0**	**54.5**	**51.81**	**+4.90**

Table 4: **Evaluation of CodeGen and LLaMA models on several text-to-SQL benchmarks**. Due to the space constraints, we only present the contrastive results of most relevant KD counterparts, *i.e.*, RKD and GKD.

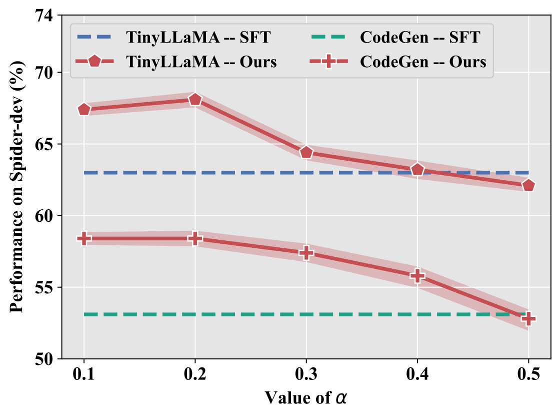

Figure 5: **Parameter analysis of masking ratio** α. We report the EX results of TinyLLaMA-1.1B and CodeGen-350M on the Spider-dev.

Method	CodeGen	TinyLLaMA
SFT	53.1	63.0
Vanilla KID	55.1	66.0
-w/ Masking-only	55.8 (↑ 0.7)	66.5 (↑ 0.5)
-w/ Rewriting (Ours)	58.4 (↑ **3.3**)	68.1 (↑ **2.1**)

Table 5: **Impact of rewriting approach of KID**. Notably, "Vanilla KID" means that we do not train with the imperfect data in our KID, "-w/ Masking-only" denotes that we directly use the sequence with masking spans as final imperfect data during the training of KID, and "-w/ Rewriting (Ours)" refers to the full KID.

tently outperforms the SFT baseline. 2) Performance of difficulty-driven strategies (*i.e.*, "Easy" and "Hard") is unstable, as paying too much attention to the easy-to-learn/hard-to-learn tokens might affect the learning of the other tokens and thus leads to sub-optimal performance. 3) The "Random" strategy achieves consistently better performance. We conjecture that such a random masking strategy is closer to the errors that are prone to occur during inference, as a model might predict incorrect tokens at any inference step. Thus, we use the "Random" strategy as our default setting.

Parameter analysis on α. The α used to control the ratio of masking tokens is an important hyper-parameter. Here, we analyze its influence by evaluating the performance of KID with different α, spanning $\{0.1, 0.2, 0.3, 0.4, 0.5\}$ on Spider-dev. Figure 5 illustrates the contrastive results. Compared with the SFT baseline, our KID consistently brings improvements across a certain range of α (*i.e.*, 0.1 to 0.3), basically indicating that the performance of KID is not sensitive to α. 2) Too large α values (*e.g.*, 0.5) lead to performance degradation, as too many rewriting tokens might distort the sequence meaning and are challenging for models to calibrate. More specifically, the case of $\alpha = 0.2$ performs best, and we use this setting as default.

Impact of rewriting approach. In the stage ❸ of pipeline for obtaining the imperfect data, we rewrite the ground-truth data with the predicted imperfect tokens. To verify its effectiveness, we compare it with a simple alternative, *i.e.*, directly using the sequence with masking spans (output of stage ❶) as final imperfect data $\hat{y}$, denoted as "-w/ masking-only". Table 5 shows the contrastive results (EX results on Spider-dev), in which we see that 1) the alternative approach equipped with KID outperforms the SFT, showing the superiority of our KID, and importantly, 2) our rewriting approach

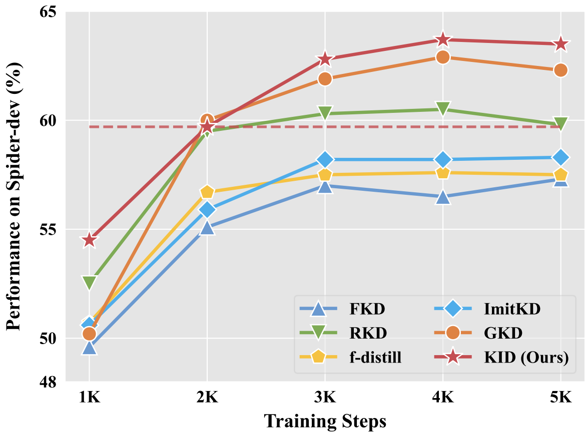

Figure 6: **Performance on Spider-dev of students (QWen1.5-0.5B) trained with different KD methods for the full training process**. QWen1.5-1.8B is used as the teacher. We see that KID achieves comparable performance with most counterparts at 2K training steps.

could further improve the results by a large margin against the simple alternative, *e.g.,* +3.3% gains on CodeGen-350M, indicating its effectiveness.

Analysis of training efficiency. In Table 3, we show that our KID effectively reduces the training latency compared to those counterparts based on model-generated data. Here, to further verify the training efficiency of KID, we present the performance of students trained with various KD methods across different training steps. QWen1.5-0.5B and 1.8B models are used as student and teacher, respectively. The results are illustrated in Figure 6. As seen, KID can achieve comparable or even better performance than most KD counterparts with much fewer training steps, *i.e.*, effectively improving the training efficiency. We attribute it to the higher data efficiency, since the imperfect data is closer to inference scenarios and can help the student better adapt to downstream generation.

5 Related Work

LLM-based Text-to-SQL. Recently, autoregressive LLMs (OpenAI, 2023; Ouyang et al., 2022; Touvron et al., 2023; Anil et al., 2023; Zhao et al., 2023) have shown their superior performance by solving various NLP tasks in a generative manner. In the field of text-to-SQL, researchers are increasingly interested in leveraging the powerful capabilities of LLMs to create text-to-SQL systems, which can be classified into two groups: 1) prompt-based text-to-SQL and training-based text-to-SQL. The former involves designing some effective prompts to instruct the closed-source LLMs for better text-to-SQL parsing (Pourreza and Rafiei, 2024; Sun et al., 2023a; Chen et al., 2024; Dong et al., 2023). On the other hand, the training-based methods aim to improve the text-to-SQL performance of open-source LLMs by tuning them on the supervised input-output pairs (Sun et al., 2023a; Zhang et al., 2024a), or continuing pretraining the LLMs on the related database-related data (Roziere et al., 2023; Li et al., 2024a). While achieving remarkable performance, the above methods usually suffer from unbearable inference latency (Zhong et al., 2024; Leviathan et al., 2023), hindering the applications in real-world scenarios.

Knowledge Distillation for Autoregressive LLMs. KD, as a common approach for compressing LLMs, has attracted great attention recently (Gu et al., 2023; Agarwal et al., 2024; Zhong et al., 2024; Xu et al., 2024). In the context of text-to-SQL, Sun et al. (2023b) is first to apply the KD for distilling the text-to-SQL models, but they mainly focus on the encoder-only (Devlin et al., 2019) and sequence-to-sequence models (Raffel et al., 2020). It still under-explored whether these methods work well for distilling the autoregressive text-to-SQL LLMs. Hence, we attempt to explore it and propose a more efficient KD method that is more suitable for text-to-SQL LLMs. To the best of our knowledge, we are one of the rare works that focus on efficient LLM-based text-to-SQL systems, and we hope our work can promote more related research in this field.

6 Conclusion

In this paper, we reveal and address the limitations of current KD methods in compressing the autoregressive text-to-SQL LLMs. Based on a series of preliminary analyses, we find that these methods fall short in balancing performance and training efficiency. To this end, we propose a novel efficient KD algorithm (KID), which utilizes a simple-yet-effective strategy to simulate the inference scenarios during training, with only a one-pass forward process. By doing so, KID can mitigate the training-inference mismatch in an efficient manner, and achieve a better trade-off between performance and efficiency. Experiments show that our approach consistently and significantly improves distillation performance across all model architectures, and reduces the training latency by a large margin.

Limitations

Our work has several potential limitations. First, given the limited computational budget, we only validate our `KID` on up to 7B LLMs in the main experiments. It will be more convincing if scaling up to super-large model size (*e.g.*, 70B) and applying `KID` to more cutting-edge model architectures. On the other hand, besides the distillation for the text-to-SQL task, we believe that our method has the great potential to expand to more scenarios, *e.g.*, distilling the general-purpose abilities of LLMs, which are not fully explored in this work.

Ethics and Reproducibility Statements

Ethics. We take ethical considerations very seriously and strictly adhere to the ACL Ethics Policy. This paper proposes an efficient knowledge distillation algorithm for text-to-SQL LLMs. It aims to compress the existing larger LLMs into smaller ones, instead of encouraging them to learn privacy knowledge that may cause the ethical problem. Moreover, all training and evaluation datasets used in this paper are publicly available and have been widely adopted by researchers. Thus, we believe that this research will not pose ethical issues.

Reproducibility. In this paper, we discuss the detailed experimental setup and provide enough information to re-product our results, such as statistic descriptions and training hyper-parameters. More importantly, ***we have provided our code in the supplementary materials*** to help reproduce the experimental results of this paper.

References

Rishabh Agarwal, Nino Vieillard, Piotr Stanczyk, Sabela Ramos, Matthieu Geist, and Olivier Bachem. 2024. On-policy distillaiton of language models: Learning from self-generated mistakes. In *ICLR*.

Rohan Anil, Andrew M Dai, Orhan Firat, Melvin Johnson, Dmitry Lepikhin, Alexandre Passos, Siamak Shakeri, Emanuel Taropa, Paige Bailey, Zhifeng Chen, et al. 2023. Palm 2 technical report. *arXiv preprint*.

Jinze Bai, Shuai Bai, Yunfei Chu, Zeyu Cui, Kai Dang, Xiaodong Deng, Yang Fan, Wenbin Ge, Yu Han, Fei Huang, et al. 2023. Qwen technical report. *arXiv preprint*.

Xinyun Chen, Maxwell Lin, Nathanael Schaerli, and Denny Zhou. 2024. Teaching large language models to self-debug. In *ICLR*.

Xiang Deng, Ahmed Hassan, Christopher Meek, Oleksandr Polozov, Huan Sun, and Matthew Richardson. 2021. Structure-grounded pretraining for text-to-sql. In *NAACL*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL*.

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

Bent Fuglede and Flemming Topsoe. 2004. Jensen-shannon divergence and hilbert space embedding. In *International symposium onInformation theory, 2004. ISIT 2004. Proceedings.*

Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R Woodward, Jinxia Xie, and Pengsheng Huang. 2021a. Towards robustness of text-to-sql models against synonym substitution. In *ACL*.

Yujian Gan, Xinyun Chen, and Matthew Purver. 2021b. Exploring underexplored limitations of cross-domain text-to-sql generalization. In *EMNLP*.

Yuxian Gu, Li Dong, Furu Wei, and Minlie Huang. 2023. Knowledge distillation of large language models. *arXiv preprint*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv preprint*.

Edward J Hu, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. 2021. Lora: Low-rank adaptation of large language models. In *ICLR*.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *The VLDB Journal*.

Yoon Kim and Alexander M Rush. 2016. Sequence-level knowledge distillation. In *EMNLP*.

Yaniv Leviathan, Matan Kalman, and Yossi Matias. 2023. Fast inference from transformers via speculative decoding. In *ICML*.

Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql. In *AAAI*.

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

Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024b. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. In *NeurIPS*.

Alexander Lin, Jeremy Wohlwend, Howard Chen, and Tao Lei. 2020. Autoregressive knowledge distillation through imitation learning. In *EMNLP*.

Xiaoxuan Liu, Lanxiang Hu, Peter Bailis, Ion Stoica, Zhijie Deng, Alvin Cheung, and Hao Zhang. 2023. Online speculative decoding. In *ICLR*.

Andrey Malinin and Mark Gales. 2019. Reverse kl-divergence training of prior networks: Improved uncertainty and adversarial robustness. *NeurIPS*.

Erik Nijkamp, Bo Pang, Hiroaki Hayashi, Lifu Tu, Huan Wang, Yingbo Zhou, Silvio Savarese, and Caiming Xiong. 2022. Codegen: An open large language model for code with multi-turn program synthesis. In *ICLR*.

OpenAI. 2023. Gpt-4 technical report. *Preprint*, arXiv preprint:2303.08774.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. In *NeurIPS*.

Richard Yuanzhe Pang and He He. 2020. Text generation by learning from demonstrations. In *ICLR*.

Mohammadreza Pourreza and Davood Rafiei. 2024. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. In *NeurIPS*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *JMLR*.

Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint*.

Roy Schwartz, Jesse Dodge, Noah A Smith, and Oren Etzioni. 2020. Green ai. *Communications of the ACM*.

Ruoxi Sun, Sercan O Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023a. Sql-palm: Improved large language modeladaptation for text-to-sql. *arXiv preprint*.

Shuo Sun, Yuze Gao, Yuchen Zhang, Jian Su, Bin Chen, Yingzhan Lin, and Shuqi Sun. 2023b. An exploratory study on model compression for text-to-sql. In *Findings of ACL*.

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint*.

Tim Van Erven and Peter Harremos. 2014. Rényi divergence and kullback-leibler divergence. *IEEE Transactions on Information Theory*.

Sergio Verdú. 2014. Total variation distance and the distribution of relative information. In *2014 Information Theory and Applications Workshop (ITA)*.

Yuqiao Wen, Zichao Li, Wenyu Du, and Lili Mou. 2023. f-divergence minimization for sequence-level knowledge distillation. In *ACL*.

Taiqiang Wu, Chaofan Tao, Jiahao Wang, Zhe Zhao, and Ngai Wong. 2024. Rethinking kullback-leibler divergence in knowledge distillation for large language models. *arXiv preprint*.

Xiaohan Xu, Ming Li, Chongyang Tao, Tao Shen, Reynold Cheng, Jinyang Li, Can Xu, Dacheng Tao, and Tianyi Zhou. 2024. A survey on knowledge distillation of large language models. *arXiv preprint*.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. In *EMNLP*.

Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024a. Benchmarking the text-to-sql capability of large language models: A comprehensive evaluation. *arXiv preprint*.

Peiyuan Zhang, Guangtao Zeng, Tianduo Wang, and Wei Lu. 2024b. Tinyllama: An open-source small language model. *arXiv preprint*.

Wayne Xin Zhao, Kun Zhou, Junyi Li, Tianyi Tang, Xiaolei Wang, Yupeng Hou, Yingqian Min, Beichen Zhang, Junjie Zhang, Zican Dong, et al. 2023. A survey of large language models. *arXiv preprint*.

Qihuang Zhong, Liang Ding, Juhua Liu, Bo Du, and Dacheng Tao. 2023. Self-evolution learning for discriminative language model pretraining. In *Findings of ACL*.

Qihuang Zhong, Liang Ding, Li Shen, Juhua Liu, Bo Du, and Dacheng Tao. 2024. Revisiting knowledge distillation for autoregressive language models. In *ACL*.

Ruiqi Zhong, Tao Yu, and Dan Klein. 2020. Semantic evaluation for text-to-sql with distilled test suites. In *EMNLP*.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint*.

Xunyu Zhu, Jian Li, Yong Liu, Can Ma, and Weiping Wang. 2023. A survey on model compression for large language models. *arXiv preprint*.

A Appendix

A.1 Details of Tasks and Datasets

In this work, we conduct extensive experiments on several text-to-SQL benchmarks. Here, we introduce the descriptions of these datasets in detail. Firstly, we present the statistics of all used datasets in Table 6. Then, each task is described as:

Spider. Spider (Yu et al., 2018) is a widely-used English text-to-SQL benchmark, comprising 8,659 training samples and 1,034 development samples. The training set encompasses 7,000 manually annotated samples and 1,659 samples sourced from six previous text-to-SQL benchmarks. There are 200 databases covering 138 diverse domains in Spider. Due to the submission constraints of the Spider leaderboard, we follow Li et al. (2024a) and do not evaluate our models on its test set, but alternatively on the publicly available development set.

BIRD. BIRD (Li et al., 2024b) is a more challenging text-to-SQL benchmark that examines the impact of extensive database contents on text-to-SQL parsing. BIRD contains over 12,751 unique question-SQL pairs and 95 big databases with a total size of 33.4 GB. Each database contains around 549K rows on average.

Spider-DK. Spider-DK (Gan et al., 2021b) is a variant derived from the original Spider dataset. It modifies some samples of Spider by adding domain knowledge that reflects real-world question paraphrases.

Spider-Realistic. Spider-Realistic (Deng et al., 2021) is also a variant of Spider dataset. It modifies the NL questions in the complex subset of Spider to remove or paraphrase explicit mentions of column names, while keeping the SQL queries unchanged.

Spider-Syn. Spider-Syn (Gan et al., 2021a) is a human-curated dataset based on the Spider. NL questions in Spider-Syn are modified from Spider, by replacing their schema-related words with manually selected synonyms that reflect real-world question para-phrases.

A.2 Training Hyper-parameters.

We train each model with a batch size of 16 and a peak learning rate of 2e-4. The training epochs are selected from {4, 8} for different models. We follow Li et al. (2024a) to construct the database prompt (an example of an input-output pair is illustrated in Figure 7) and set the max length of input and output depending on different models. Due to the limited computational resources, we train all models with a popular parameter-efficient fine-tuning method, *i.e.*, LoRA. Specifically, the alpha of LoRA is set as 32 and the rank of LoRA is set as 64 or 8. We present the training hyper-parameters in Table 7. All experiments are conducted on 8 NVIDIA H800 (80GB) GPUs.

Benchmark	#Training	#Development
Spider	8,659	1,034
BIRD	9,428	1,534
Spider-DK	-	535
Spider-Realistic	-	508
Spider-Syn	-	1,034

Table 6: **Statistic of all used text-to-SQL benchmarks**. Notably, "Spider-DK", "Spider-Realistic" and "Spider-Syn" are variants of the development of Spider.

Setting	QWen1.5	CodeGen	LLaMA2
Learning Rate	2e-4	2e-4	2e-4
Epoch	8	8	4
Batch Size	16	16	16
Max Input Length	1024	1024	2048
Max Output Length	128	128	256
LoRA_Rank	64	8	64
LoRA_Alpha	32	32	32

Table 7: **Details of training hyper-parameters for different LLMs**. For each model, we use the same settings among all benchmarks.

A.3 Details of divergence functions for KD

Here, we introduce the commonly-used divergence functions for KD. Let the probability distribution of teacher and student be p and q^θ, respectively. For the training set $\mathcal{G}$, the divergence functions can be formulated as:

Kullback-Leibler (KL) divergence

$$\mathcal{F}_{KL}(p\|q^\theta) = \sum_{(x,y)\in\mathcal{G}} p(y|x) \log \frac{p(y|x)}{q^\theta(y|x)}. \quad (3)$$

Note that the KL divergence is not symmetric, *i.e.*, $\mathcal{F}_{KL}(p\|q^\theta) \neq \mathcal{F}_{KL}(q^\theta\|p)$. More specifically, the $\mathcal{F}_{KL}(p\|q^\theta)$ refers to the forward KL, while $\mathcal{F}_{KL}(q^\theta\|p)$ refers to the reverse KL.

Jensen–Shannon (JS) divergence

$$\mathcal{F}_{JS}(p\|q^\theta) = \frac{1}{2}(\mathcal{F}_{KL}(p\|M) + \mathcal{F}_{KL}(q^\theta\|M)), \quad (4)$$

where $M = \frac{1}{2}(p + q^\theta)$.

INPUT

Database prompt:
table **movie** , columns = [movie.mid (int | primary key | comment : movie id | values : 101 , 102) , movie.title (text | values : Gone with the Wind , Star Wars) , movie.year (int | values : 1939 , 1977) , movie.director (text | values : Victor Fleming , George Lucas)]
table **reviewer** , columns = [reviewer.rid (int | primary key | comment : reviewer id | values : 201 , 202) , reviewer.name (text | values : Sarah Martinez , Daniel Lewis)]
table **rating** , columns = [rating.rid (int | comment : reviewer id | values : 201 , 202) , rating.mid (int | comment : movie id | values : 101 ,106) , rating.stars (int | comment : rating stars | values : 2 , 4) , rating.ratingdate (date | values : 2011-01-22 , 2011-01-27)]
foreign keys :
rating.rid = reviewer.rid
rating.mid = movie.mid
matched values :
reviewer.name (Sarah Martinez)
Question:
What are the names of all directors whose movies have been reviewed by Sarah Martinez?

OUTPUT

SELECT DISTINCT movie.director **FROM** rating **JOIN** movie **ON** rating.mid = movie.mid **JOIN** reviewer **ON** rating.rid = reviewer.rid **WHERE** reviewer.name = 'Sarah Martinez'

Figure 7: **A text-to-SQL sample in Spider's training set**. We follow Li et al. (2024a) to construct the database prompts. Note that this illustration is from the original paper (Li et al., 2024a).

Total variation distance (TVD)

$$\mathcal{F}_{TVD}(p||q^{\theta}) = \sum_{(x,y)\in\mathcal{G}} |\frac{p(y|x) - q^{\theta}(y|x)}{2}|. \quad (5)$$