

MSc-SQL: Multi-Sample Critiquing Small Language Models For Text-To-SQL Translation

Satya Krishna Gorti, Ilan Gofman, Zhaoyan Liu, Jiapeng Wu,
Noël Vouitsis, Guangwei Yu, Jesse C. Cresswell, Rasa Hosseinzadeh,
Layer 6 AI

{satya, ilan, zhaoyan, paul, noel, guang, jesse, rasa}@layer6.ai

Abstract

Text-to-SQL generation enables non-experts to interact with databases via natural language. Recent advances rely on large closed-source models like GPT-4 that present challenges in accessibility, privacy, and latency. To address these issues, we focus on developing small, efficient, and open-source text-to-SQL models. We demonstrate the benefits of sampling multiple candidate SQL generations and propose our method, MSc-SQL, to critique them using associated metadata. Our sample critiquing model evaluates multiple outputs simultaneously, achieving state-of-the-art performance compared to other open-source models while remaining competitive with larger models at a much lower cost. Full code can be found at github.com/layer6ai-labs/msc-sql.

1 Introduction

Text-to-SQL generation is a rapidly growing area of natural language processing with significant real-world applications. It enables non-expert users to interact with databases using natural language queries, which are then automatically translated into SQL queries. This capability is widely applicable across various domains, including business intelligence, customer service automation, and data analysis. For example, it facilitates automating the feature selection process in tabular data prediction tasks which often involves complex aggregation queries over event histories. Furthermore, empowering translation models with database query access will allow for automated agentic workflows.

Recent advances in text-to-SQL generation have primarily leveraged closed-source models like GPT-4 (OpenAI, 2023) which, combined with advanced prompting techniques, have consistently achieved state-of-the-art performance on benchmarks such as Spider (Yu et al., 2018) and BIRD (Li et al., 2023b). However, the reliance on closed-source API-based models limits accessibility, trans-

parency, and task adaptability, while presenting privacy concerns around data being sent to an API. All of these limitations underscore the need for efficient, open-source alternatives capable of competitive performance.

The inherent complexity of text-to-SQL tasks has contributed to proposed solutions becoming increasingly expensive. For instance, recent research (Wang et al., 2023; Pourreza and Rafiei, 2024; Talaei et al., 2024) demonstrates that decomposing complex tasks into specialized sub-tasks – such as table prediction, SQL generation, and error correction – improves overall performance. However, these composite approaches also increase the number of function calls to the model, leading to latency issues, especially when relying on larger models. This highlights the importance of developing smaller, task-specific models that maintain accuracy while being computationally efficient. Efforts such as DTS-SQL (Pourreza and Rafiei, 2024) and SFT CodeS-7B (Li et al., 2024a) are among the few that try to address this need.

Our objective is to develop efficient methods for text-to-SQL generation that succeed with small and open-source models. We demonstrate that smaller language models (under 10B parameters) struggle to match the performance of their larger closed-source counterparts when relying solely on existing approaches, namely those that combine schema linking with SQL generation. We then show that this gap can be closed by sampling and running multiple SQL queries – either from the same model or from an ensemble of models of similar size – and comparing the results. By limiting the number of samples to two or three, this strategy balances between improving generation quality and maintaining computational efficiency. Recent works on other natural language generation tasks (Brown et al., 2024; Snell et al., 2024; Li et al., 2024b) support the idea that increased test-time compute can boost generation quality.

Sampling multiple SQL queries requires a way to judge the various candidates and then select the best one. Prior works on natural language generation with Large Language Models (LLMs) have employed techniques like training a ranker to evaluate the quality of generated samples (Li et al., 2022a), or using reward models to guide the selection process (Ouyang et al., 2022; Rafailov et al., 2023). However, in the more specialized context of text-to-SQL generation we propose a distinct approach: training a sample-critiquing model that simultaneously considers multiple generations, their corresponding execution results, and associated metadata to determine which of the generated SQL queries should be returned. Compared to analogous methods, our approach allows the model to better leverage comparative information and contextual cues. Our results show state-of-the-art performance among open-source models on popular text-to-SQL benchmarks, while also achieving competitive results against larger closed-source models albeit at a much lower cost.

2 Related Work

Text-to-SQL Generation. Early works in the space of text-to-SQL generation predominantly leveraged rule-based methods to parse natural language queries and map them to valid SQL statements (Popescu et al., 2003, 2004; Li and Jagadish, 2014). Recently, LLMs such as GPT-4 (OpenAI, 2023) have facilitated this task by leveraging their strong textual priors to generate SQL queries (Li et al., 2023b). Due to the challenging nature of direct SQL generation, subsequent works instead decomposed the generation process into dedicated sub-tasks and prompted GPT-4 to solve each task sequentially (Wang et al., 2023; Dong et al., 2023; Pourreza and Rafiei, 2023; Chen et al., 2024). For example, MAC-SQL (Wang et al., 2023) defined three sub-components namely a Selector, a Decomposer, and a Refiner, where dedicated prompts were engineered for each component. Despite being state-of-the-art on several SQL generation benchmarks, the reliance of these methods on very large and closed-source models makes them inherently inefficient and expensive, and poses accessibility and privacy concerns. More recently, a few works have begun exploring the use of smaller open-source models (under 10 billion parameters) for SQL generation tasks (Pourreza and Rafiei, 2024; Li et al., 2024a) and have shown promising re-

sults. However, there remains a significant performance gap compared to several of the aforementioned GPT-4-based models. Our method MSc-SQL largely bridges this accuracy gap while remaining efficient and open-source.

Exploring Test-Time Computation. On general LLM benchmarks, recent methods have sought to leverage additional test-time computation, such as planning, reasoning and problem deliberation to improve performance (Wei et al., 2022; Yao et al., 2023b; Besta et al., 2024; Zelikman et al., 2024; Yao et al., 2023a). For example, Chain-of-Thought (CoT) prompting (Wei et al., 2022) forces the model to spend more tokens “thinking” about the problem before answering. Some methods have also explored repeated sampling as a way of using increased test-time computation to expand the space of generated solutions (Cobbe et al., 2021; Irvine et al., 2023; Snell et al., 2024; Brown et al., 2024; Zhang et al., 2024a). This expanded solution set is then filtered out using either rule based verifiers (Wang et al., 2022), such as picking the response that passes all the test-cases in the coding domain, or using other models to compare and then select the best sample among candidates (Cobbe et al., 2021; Lightman et al., 2024). We explore the latter direction in the domain of text-to-SQL generation.

3 Towards Building an Efficient Text-to-SQL Pipeline

Problem Setup. Text-to-SQL generation involves translating a natural language query q into a structured SQL query s that retrieves the desired information from a database. More formally, given a natural language question q , the schema \mathcal{S} of a relational database \mathcal{D} , and any extra metadata \mathcal{M} associated with the query such as extra evidence or few-shot examples from the database, the goal is to generate a valid SQL query s such that the execution of s on \mathcal{D} returns the correct answer to q . In other words, the overall objective is to learn a model $f : (q, \mathcal{S}, \mathcal{M}) \rightarrow s$. This problem is challenging on many fronts because of the diversity in which the natural language question can be expressed, the difficulty in inferring complex relational structures from a database schema, and the restriction of generating only valid SQL syntax for s . Due to these complexities, we divide the problem into three distinct modules which form a blueprint for an efficient text-to-SQL pipeline,

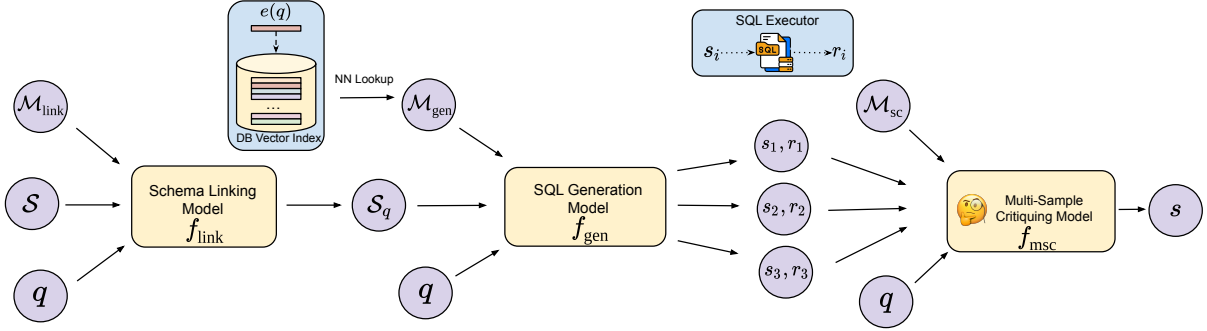


Figure 1: Starting with a natural language query q , database schema \mathcal{S} , and metadata $\mathcal{M}_{\text{link}}$, the schema linking model returns a subset \mathcal{S}_q of tables which are necessary to answer q . Next, the SQL generation model adds metadata \mathcal{M}_{gen} obtained through retrieval against an embedding of the query $e(q)$, and generates multiple possible SQL queries s_i . Finally, the multi-sample critiquing model comparatively evaluates the generations s_i along with their execution results r_i when run on the database, and then selects one as the final output s .

depicted in Figure 1. While the first two modules are commonly used, we make novel design recommendations in our blueprint. The third module described in section 4 is original.

3.1 Schema Linking

Schema linking is the task of identifying the relevant tables and attributes within the schema \mathcal{S} that are necessary to construct a valid SQL query s based on the natural language query q . This step is critical because SQL queries typically involve only a subset of the available tables and attributes. Correctly identifying this subset ensures that the subsequent steps including SQL generation are focused on the most pertinent tables in the schema. Schema linking also reduces the input’s length to help it fit in the limited context size of existing language models, while reducing costs in the quadratic Transformer attention (Vaswani et al., 2017) operation, and improving context utilization (Liu et al., 2024). Given the schema \mathcal{S} containing information about each table and column type including the primary key and foreign key relationships, and corresponding metadata $\mathcal{M}_{\text{link}}$, we denote our schema linking model as $f_{\text{link}} : (q, \mathcal{S}, \mathcal{M}_{\text{link}}) \rightarrow \mathcal{S}_q$, where $\mathcal{S}_q \subseteq \mathcal{S}$ represents only the schemas of the tables that are predicted to be needed for answering q . Our aim here is to maximize the recall of predicted tables; high recall is essential because missing even one necessary table immediately precludes complete and correct SQL generation. While emphasizing recall may introduce some false positives (i.e., irrelevant tables included in \mathcal{S}_q), our subsequent stages are designed to be robust to such inaccuracies.

3.2 SQL Generation

Given a reduced schema \mathcal{S}_q , we then proceed to generate a SQL query s as $f_{\text{gen}} : (q, \mathcal{S}_q, \mathcal{M}_{\text{gen}}) \rightarrow s$, where \mathcal{M}_{gen} is associated metadata. Although conceptually simple, generating a valid SQL query often requires knowledge of the formatting of column values, for instance when using a WHERE clause. To generate a correct SQL query s for the example q “Which school is in California?”, f_{gen} must know whether the state of California is represented as “CA”, “California”, or another variant thereof in the particular SQL database \mathcal{D} .

Contextual Retrieval through Few-Shot Examples. We thus augment the SQL generation process by providing few-shot examples of the values in each column as additional metadata in \mathcal{M}_{gen} . By including this information, the model can better infer the correct SQL query, resolving potential ambiguities related to data representation in the database. For string columns it is especially important to provide information which is relevant to the query; hence we use the nearest neighbours of q . The retrieval of few-shot examples is conducted through a similarity measure with an embedding of the input query concatenated with any additional evidence provided as metadata. A vector index on the entire database is constructed offline by indexing a fixed number of unique entries of every string type column. We specifically use Alibaba-NLP/gte-large-en-v1.5 (Zhang et al., 2024b) as the embedding model. For other column types, we randomly sample few-shot examples.

Robust Training with Noisy Table Injection. The schema linking stage may predict more tables than

are strictly necessary in \mathcal{S}_q because it aims to maximize recall. To combat this, we improve the robustness of generation to noise in \mathcal{S}_q by having the SQL generation step learn to discard unnecessary and irrelevant tables. Starting with the ground-truth schema \mathcal{S}_q^* from the training set containing only necessary tables needed for query q , we inject extra tables denoting the result by \mathcal{S}_q^\dagger . We sample anywhere between 0 to 2 extra tables with a weighted probability, then train f_{gen} on \mathcal{S}_q^\dagger rather than \mathcal{S}_q^* .

Training. Starting with an open-source language model, we fine-tune on tuples $(q, \mathcal{S}_q^\dagger, \mathcal{M}_{\text{gen}}, s)$ so that the model learns to generate syntactically correct and semantically valid SQL queries. The generation follows a sequence to sequence paradigm to maximize the likelihood of outputs,

$$\mathcal{L}_{\text{gen}} = - \sum_{(q, \mathcal{S}_q^\dagger, \mathcal{M}_{\text{gen}}, s)} \log P(s | q, \mathcal{S}_q^\dagger, \mathcal{M}_{\text{gen}}), \quad (1)$$

where P represents the probability assigned to s by the language model f_{gen} . Integrating both retrieval of contextual examples through \mathcal{M}_{gen} and exposure to superfluous tables through \mathcal{S}_q^\dagger refines the model’s capability to discern relevant schema information, enhancing the accuracy of the generated SQL queries.

4 MSc-SQL: Enhancing SQL Generation Using Multi-Sample Critiquing

As mentioned in [section 1](#), our objective is two-fold. We propose to use only open-source language models for their improved accessibility, transparency, adaptability, and privacy. Second, we aim to achieve high-quality SQL generation using smaller language models to promote faster inference and reduce overall computational costs. However, small language models typically lag behind their larger counterparts in generation quality due to inherent limitations in their capacity. This is especially true in challenging tasks like SQL generation that require a deep understanding of the complex hierarchies present in an SQL statement. We reconcile this gap by increasing the test-time computation budget used by smaller language models, sampling multiple generations to improve the likelihood of generating a correct SQL statement. In this work, we demonstrate that generating as few as three samples can yield highly competitive results on SQL benchmarks while maintaining the overall efficiency of the pipeline.

However, sampling multiple generations leaves us with the task of selecting the best possible generation among them. Prior works have used methods such as majority voting ([Wang et al., 2022](#)), use of a reward model ([Christiano et al., 2017](#); [Cobbe et al., 2021](#); [Lightman et al., 2024](#); [Li et al., 2022a](#)) or a combination of the two to pick the best generated candidate ([Brown et al., 2024](#)). This has shown promising results in other domains like math and coding tasks ([Hendrycks et al., 2021](#); [Li et al., 2022b](#)). Other methods use pre-trained LLMs like GPT-4 to judge the quality of the sampled generations to select the best one ([Zheng et al., 2023](#); [Lee et al., 2024](#)). Such methods have the drawback of utilizing expensive closed models like GPT-4 to evaluate the best candidate. We mitigate this cost by developing an open model for the purpose of sample critiquing.

4.1 Multi-Sample Critiquing in MSc-SQL

Sample critiquing involves evaluating the inputs and the generated samples to determine the correctness of the generation. To facilitate this decision we can increase the contextual information provided to the model so it can make a more informed critique. To this end, we provide the model with not only the question, schema and the generated SQL queries, but also with the output of executing the SQL query and any resulting error messages. For example, if the query asks to find the average employee salary grouped by department, and a generated SQL query returns a single column of average salary without the grouping column, then the critiquing model should be able to identify the discrepancy and rule out the generated query.

We let s_i and r_i represent the candidate SQL queries and their respective execution results, and let \mathcal{M}_{sc} denote any additional metadata needed for this stage. As a baseline, we first consider a basic critiquing model, using f_{isc} to denote *independent sample critiquing*, where $f_{\text{isc}} : (q, \mathcal{S}_q, (s_i, r_i), \mathcal{M}_{\text{sc}}) \rightarrow [0, 1]$. This model independently critiques the candidates s_i which are then ranked according to the model’s logits to output a single option. Although augmenting the input of f_{isc} with r_i improves the model’s ability to select the correct SQL query, by independently ranking each SQL query the model still lacks the ability to jointly reason over multiple samples to consider relative differences among them; we expect that critiquing multiple samples at once should allow the model to consider subtle nuances that might

Question	Generated SQL 1				
<p>Question: Who is the illustrator that illustrated the least amount of cards? List the format of play of the cards that he/she illustrated.</p> <p>Evidence: format of the cards refers to format; illustrator refers to artist; the least amount of cards refers to MIN(artist)</p> <p>DB schema: card_games</p> <pre>TABLE "cards" (id INTEGER primary key, -- Ex: 41138, 1349, 23810 artist TEXT, -- artist that illustrated the card art. Ex: "Pete", "Andrea Radeck", "Illustranesia" ...) TABLE "legalities" (id INTEGER primary key, -- unique id legality. Ex: 1, 2, 3 format TEXT, -- format of play. Ex: "pauper", "standard", "penny" ... foreign key (uuid) references cards(uuid)) ...</pre>	<pre>SELECT T2.artist, T1.format FROM legalities AS T1 INNER JOIN cards AS T2 ON T1.uuid = T2.uuid GROUP BY T2.artist ORDER BY COUNT(T2.id) ASC LIMIT 1</pre> <p style="text-align: center;">Result of SQL 1</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>artist</th> <th>format</th> </tr> </thead> <tbody> <tr> <td>Anna Pavleeva</td> <td>gladiator</td> </tr> </tbody> </table> <p style="text-align: right; color: green; font-size: 2em;">✓</p>	artist	format	Anna Pavleeva	gladiator
	artist	format			
Anna Pavleeva	gladiator				
<pre>SELECT T1.artist, T2.format FROM cards AS T1 INNER JOIN legalities AS T2 ON T1.uuid = T2.uuid GROUP BY T1.artist, T2.format ORDER BY COUNT(T1.artist) LIMIT 1</pre> <p style="text-align: center;">Result of SQL 2</p> <table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th>artist</th> <th>format</th> </tr> </thead> <tbody> <tr> <td style="background-color: #f2dede;">None</td> <td>commander</td> </tr> </tbody> </table> <p style="text-align: right; color: red; font-size: 2em;">✗</p>	artist	format	None	commander	
artist	format				
None	commander				

Figure 2: Two example queries sampled from our SQL generation model. Both are given to MSc-SQL for critiquing; one is correct and one is incorrect. Joint reasoning over both queries allows MSc-SQL to better capture the nuanced differences between them and thus select the correct query.

make one query correct over another.

We therefore propose to simultaneously critique multiple generated samples (provided they fit within the maximum context size of the underlying model) to allow for a more comprehensive comparison. We illustrate an example in Figure 2 where we provide two generated SQL samples that MSc-SQL must choose between. The question asks to identify the illustrator with the least amount of cards, along with clarifying evidence on the ‘format’ and ‘artist’ columns. While the first generated query is correct, the second one wrongly groups the ‘artist’ and ‘format’ columns resulting in the value *None* for the ‘artist’ column (highlighted in red in the figure). Providing the model with both generations at once makes it easier to discern the correct response. We only illustrate two generated samples for clarity, but MSc-SQL generalizes to n candidate samples as $f_{\text{msc}} : (q, \mathcal{S}, \{(s_i, r_i)\}_{i=1}^n, \mathcal{M}_{\text{sc}}) \rightarrow \{1, \dots, n\}$ where the output is now the index of the selected SQL sample.

Training. To facilitate training of our critiquing model and leverage the existing knowledge embedded in pre-trained open-source models, we model this task in a similar manner to next token prediction and simply predict the correct sample index as the next token.

4.2 Inference

The end-to-end inference process is specified in Algorithm 1. In cases where the context size of the underlying model is not big enough to capture all the candidates, we can reduce the context size by employing pairwise comparisons on all pairs, a sliding window strategy (Sun et al., 2023), or a tournament-style voting mechanism where candi-

Algorithm 1: Inference with MSc-SQL

Input: Natural language query q , schema \mathcal{S} , metadata $\mathcal{M}_{\text{link}}, \mathcal{M}_{\text{gen}}, \mathcal{M}_{\text{sc}}$

Output: Generated SQL query s

Step 1: Schema Linking

Predict relevant schema subset:

$$\mathcal{S}_q = f_{\text{link}}(q, \mathcal{S}, \mathcal{M}_{\text{link}})$$

Step 2: SQL Generation

Retrieve nearest-neighbor examples and enrich metadata \mathcal{M}_{gen}

Generate candidates:

$$s_i = f_{\text{gen}}(q, \mathcal{S}_q, \mathcal{M}_{\text{gen}}), \quad i = 1, \dots, n$$

Step 3: Sample Critiquing

Execute candidates to obtain results r_i

Select the best candidate:

$$i^* = f_{\text{msc}}(q, \mathcal{S}_q, \{(s_i, r_i)\}_{i=1}^n, \mathcal{M}_{\text{sc}})$$

Return Selected SQL query $s = s_{i^*}$

date pairs are arranged in a tournament bracket to reduce the number of comparisons required. Overall, we find that MSc-SQL yields highly competitive results, outperforming several methods that use proprietary LLMs like GPT-4, and stands as the best performing open-source model for text-to-SQL generation.

5 Experiments

Implementation Details. For our experiments, we considered open-source language models with fewer than 10 billion parameters. Specifically, we used instruction-tuned variants of Mistral-7B-v0.3 (Jiang et al., 2023), Llama-3-8B (Dubey et al., 2024) and Gemma-2-9B (Riviere et al., 2024) models. Fine-tuning was performed using QLoRA

Table 1: Results comparing recent methods on the BIRD and Spider benchmarks. Params. refers to the number of model parameters; if multiple models are used we select the single largest.

Method	Params.	Dev EX%	Method	Params.	Test EX%
Closed Proprietary Models			Closed Proprietary Models		
Distillery+GPT-4o-finetune (Maamari et al., 2024)	NA	67.2	CHES+GPT-4 (Talaie et al., 2024)	NA	87.2
Distillery+Gemini-1.5-Pro (Maamari et al., 2024)	NA	60.5	DAIL-SQL+GPT-4 (Gao et al., 2024)	NA	86.6
CHES+GPT-4 (Talaie et al., 2024)	NA	65.0	DIN-SQL+GPT-4 (Pourreza and Rafiei, 2023)	NA	85.3
MCS-SQL+GPT-4 (Lee et al., 2024)	NA	63.4	C3+ChatGPT (Dong et al., 2023)	NA	82.3
MAC-SQL+GPT-4 (Wang et al., 2023)	NA	59.6	Open Models		
Open Models			Open Models		
CHES+Llama-3 (Talaie et al., 2024)	70B	61.5	RESDSQL (Li et al., 2023a)	3B	79.9
Distillery+Llama-3 (Maamari et al., 2024)	405B	59.9	NatSQL+T5 (Rai et al., 2023)	3B	78.0
SFT CodeS (Li et al., 2024a)	15B	58.5	DTS-SQL+Mistral (Pourreza and Rafiei, 2024)	7B	77.0
SFT CodeS (Li et al., 2024a)	7B	57.2	MSc-SQL (Ours)	9B	84.7
DTS-SQL+DeepSeek (Pourreza and Rafiei, 2024)	7B	55.8			
MSc-SQL (Ours)	9B	65.6			

(a) BIRD Benchmark

(b) Spider Benchmark

(Dettmers et al., 2023), with a LoRA rank of 32, a LoRA α of 128, and a dropout rate of 0.05. Fine-tuning was conducted with an effective batch size of 12, using a single NVIDIA A6000 Ada GPU with 48GB VRAM. The use of QLoRA allowed for memory efficient fine-tuning of these models without the need to update all the model parameters, in line with our efficiency goal.

Datasets and Metrics. We utilized two primary datasets of text-to-SQL examples, Spider 1.0 (Yu et al., 2018) and BIRD (Li et al., 2023b), each serving distinct roles in evaluating the effectiveness of our approach to text-to-SQL generation. Spider 1.0 is a widely recognized dataset containing queries across 138 different domains spanning 200 databases. The Big Bench for Large-scale Database Grounded Text-to-SQL Evaluation (BIRD) is a more comprehensive and challenging benchmark compared to Spider 1.0. BIRD contains over 12,751 unique question-SQL pairs on 95 big databases with a total size of 33.4 GB. All of the results in our paper are reported on Spider 1.0’s test set and BIRD’s Dev set.

Key evaluation metrics for text-to-SQL generation are: Execution Accuracy (EX), Exact Match (EM), and Validity and Efficiency Score (VES) (Yu et al., 2018; Li et al., 2023b). EX measures the correctness of the SQL queries by checking if their execution results match the expected outcomes, making it a direct indicator of practical usability. EM assesses syntactic precision by comparing the generated SQL query against the reference query character by character; however, since different SQL queries can produce the same results, EM may pun-

ish functionally correct queries. VES evaluates both the correctness and computational efficiency of generated queries, which may be important in practical real-time systems.¹ Still, generating correct queries is a challenging enough problem in its own right, so we prioritize EX and forgo evaluations of syntactic precision or efficiency.

Comparison with SoTA on BIRD and Spider 1.0 Datasets. In Table 1 we provide detailed evaluation of our model’s performance against the current state-of-the-art (SoTA) methods across both major benchmarks. The results are grouped under two headings to emphasize the performance disparity between closed proprietary models and open models. Our method, MSc-SQL which integrates multi-sample critiquing, demonstrates impressive performance improvements across the two benchmarks compared to existing methods using open models. MSc-SQL critiques one sample from each of a fine-tuned Mistral-7B, Llama-8B, and Gemma-8B model to select the best candidate. Although, it only uses models under 10 billion parameters, our overall pipeline achieves a competitive score often outperforming methods that use proprietary language models such as Gemini-Pro (Reid et al., 2024) and GPT-4 (OpenAI, 2023), while also maintaining a significant advantage in inference speed. Compared to methods using open models, MSc-SQL outperforms the existing methods by a significant margin of 4.18 percentage points on BIRD.

Ablations. We show in Table 2 detailed ablation studies on different settings of our pipeline to un-

¹Note that the run-time efficiency of the generated query is separate from the efficiency of the text-to-SQL method.

Table 2: Ablations of our method on BIRD. Superscripts M , L , and G refer to the fine-tuned versions of Mistral, Llama-3, and Gemma-2 models respectively.

Model Family	Params.	Dev EX%
f_{gen}^M	7B	56.4
f_{gen}^L	8B	54.1
f_{gen}^G	9B	55.0
$f_{\text{link}}^M + f_{\text{gen}}^M$	7B	61.3
$f_{\text{link}}^M + f_{\text{gen}}^L$	8B	60.0
$f_{\text{link}}^M + f_{\text{gen}}^G$	9B	60.8
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^M\} \rightarrow f_{\text{msc}}^M$	7B	64.0
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^L, f_{\text{gen}}^L\} \rightarrow f_{\text{msc}}^M$	8B	63.5
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^G, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$	9B	62.7
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^M, f_{\text{gen}}^M\} \rightarrow f_{\text{msc}}^M$	7B	64.8
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^L, f_{\text{gen}}^L, f_{\text{gen}}^L\} \rightarrow f_{\text{msc}}^M$	8B	64.1
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^G, f_{\text{gen}}^G, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$	9B	62.9
$f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^L, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$	9B	65.6

derstand how various parts of it contribute to the performance on the BIRD benchmark. We denote the underlying LLM using a superscript: f^M , f^L , f^G denote the Mistral, Llama and Gemma models mentioned above. We first measure the effect of directly predicting SQL using f_{gen} , and then add schema linking before generation as $f_{\text{link}} \rightarrow f_{\text{gen}}$. Schema linking increases the overall accuracy by 5-6% across generation models by removing redundant tables and improving focus. We then incorporate multi-sample critiquing by using either two or three samples from the underlying generation models and let the critiquing model f_{msc} pick the most accurate SQL generation. Training such a model to critique the generations consistently increases the overall score across various settings as the sample size increases. Diversity of samples is also important. Sampling from each of a fine-tuned Mistral, Llama, and Gemma model results in the highest performance, achieving an accuracy of 65.6% on the BIRD Dev set. We denote this setting as ‘‘MSc-SQL’’ in Table 1.

We further perform analysis on the effect of using multiple generation models for sampling SQL outputs and contrast it with sampling multiple SQL outputs from a single generation model with non-zero temperature. Towards this, we train a number of different SQL generation models f_{gen} from a base Mistral model with different random seeds, obtain one SQL sample from each model (with temperature set to zero), and use our critiquing model to pick the best candidate. We plot performance on the BIRD Dev set in Figure 3 and see that the improvement in accuracy saturates between three and five models. Limiting the number of samples

Figure 3: Effect of using different models to each create one sample for multi-sample critiquing. The generation models are all fine-tuned Mistral-7B models, but with different random seeds used during training.

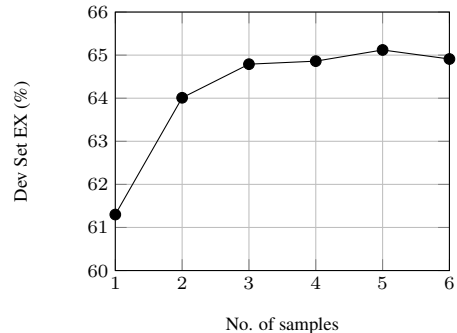


Table 3: EX% on the BIRD Dev set for varied temperatures T and number of queries used for multi-sample critiquing. The underlying generation model is a fixed fine-tune of Mistral-7B.

Samples	$T = 0$	$T = 0.1$	$T = 0.5$	$T = 1$
1	61.3	61.4	59.9	56.5
2	61.3	61.6	61.4	59.3
3	61.3	61.8	61.9	60.6
4	61.3	62.0	62.3	61.4
5	61.3	62.0	62.4	62.0

from a smaller language model to three maintains the overall efficiency of the pipeline compared to using very large models. Since these models are trained using QLoRA, the memory footprint of using multiple versions is comparable to just loading the base model.

To quantify the effect of varying temperature while sampling from a single generation model f_{gen} , we fix the temperature T to a value in $[0, 1]$, generate between one and five queries, and measure the performance of multi-sample critiquing on the BIRD Dev set with results shown in Table 3. Of course, when T is fixed to zero, sampling is deterministic and there is no benefit to critiquing. Increasing the number of samples and selecting the final output with f_{msc} consistently increases the accuracy across different T values. We note the best results with temperature $T = 0.5$.

Based on these ablations, it is evident that sampling from diverse generation models and incorporating our multi-sample critiquing model increases the overall accuracy of the text-to-SQL pipeline. Compared to sampling from a single generation model with non-zero temperature, training different models from random initializations increases the likelihood of generating a correct query which can be picked out by multi-sample critiquing, and translates to higher performance. Both abla-

Table 4: Performance of methods as the complexity of BIRD queries is varied (EX%).

	Simple	Moderate	Challenging	Overall
GPT-4-turbo	54.3	35.2	41.7	46.3
SFT CodeS-7B	64.6	46.9	40.3	57.2
SFT CodeS-15B	65.8	48.8	42.4	58.5
MAC-SQL+GPT-4	65.7	52.7	40.3	59.4
CHES+GPT-4	65.4	64.8	58.3	64.6
MSc-SQL (Ours)	72.0	58.0	49.0	65.6

tions highlight that diversity of samples is beneficial. Duplicate generated queries are not helpful in multi-sample critiquing, whereas generating diverse queries allows f_{mcs} to contrast several approaches to the problem as well as execution results. Ultimately, only a single correct query is needed, but diverse samples provide more information to determine which may be correct.

Query Complexity. In Table 4 we compare how varying query complexity affects accuracy, using annotations from the BIRD benchmark. We see an overall improvement compared to most methods on all three categories. Importantly, we observe that MSc-SQL performs much better on simple queries compared to all the current methods, while CHES+GPT-4 (Talaei et al., 2024) performs better on moderate and challenging categories. We believe this to be due to their use of GPT-4 to generate SQL queries which is expected to be much better at generating coherent complex sequences. Due to the plug-and-play nature of MSc-SQL where we can easily fine-tune and swap models in the pipeline, as smaller models improve in their ability to generate more complex sequences, the overall accuracy of our method can improve accordingly.

Analyzing Multi-Sample Critiquing. The data presented in Table 5 highlights the effectiveness of the sample critiquing methodologies described in section 4. We compare these methods with an oracle critiquing model, f_{oracle} , that always chooses the correct query when there is at least one correctly generated SQL query in the available samples. With the oracle model, the pipeline’s accuracy is 71.4% on the BIRD Dev set, representing the ceiling in terms of the performance for a fixed generation model.

To show the efficacy of the multi-sample critiquing method with f_{mcs} , we first compare two different critiquing models that process samples independently to generate a likelihood of correctness. One is an off-the-shelf high performing Llama-8B reward model taken from the Re-

Table 5: Measuring the effect of different sample critiquing techniques on BIRD.

Method	Dev EX%
Baseline: $f_{link}^M \rightarrow f_{gen}^M$	61.3
$f_{link}^M \rightarrow \{f_{gen}^M, f_{gen}^L, f_{gen}^G\} \rightarrow f_{oracle}$	71.4
$f_{link}^M \rightarrow \{f_{gen}^M, f_{gen}^L, f_{gen}^G\} \rightarrow f_{isc}^1$	60.2
$f_{link}^M \rightarrow \{f_{gen}^M, f_{gen}^L, f_{gen}^G\} \rightarrow f_{isc}^2$	62.9
$f_{link}^M \rightarrow \{f_{gen}^M, f_{gen}^L, f_{gen}^G\} \rightarrow f_{consistency}$	62.6
$f_{link}^M \rightarrow \{f_{gen}^M, f_{gen}^L, f_{gen}^G\} \rightarrow f_{msc}$	65.6

wardBench leaderboard (Lambert et al., 2024), Skywork-Reward-Llama-3.1-8B, that is tasked to predict the correctness of the generated output one at a time, with the highest ranked sample taken as the output. We refer to this model as f_{isc}^1 . We also fine-tune a Llama-8B on the same dataset as we train f_{mcs} , but with only a single query in context as described in section 4, and choose the highest ranked result. We refer to this as f_{isc}^2 . Additionally, we evaluate a third approach based on self-consistency (Wang et al., 2022), which selects the most consistent output among the generated outputs, referred as $f_{consistency}$. We see that f_{isc}^1 performs worse than the baseline (i.e. no critiquing) which is likely due to the reward model not being specifically trained on SQL tasks. While we see an improvement of 1.6 p.p. from training f_{isc}^2 on SQL data. The self-consistency approach $f_{consistency}$ shows 1.3 p.p. improvements over the baseline, comparable to f_{isc}^2 . Multi-sample critiquing f_{mcs} outperforms all other selection methods by a large margin; an improvement of 4.3 p.p. over the baseline and 2.7 p.p. over f_{isc}^2 , further demonstrating the benefits of critiquing multiple samples at once.

6 Conclusion

We present an approach for text-to-SQL translation that leverages open-source language models to build an efficient and high performing method, and show comprehensive evaluations on two existing benchmarks along with analysis on various design choices made in our approach. We show that multi-sample critiquing is needed to address the limitations of smaller language models to compete with larger and proprietary counterparts. Our critiquing model is trained to leverage richer contextual information, including query execution results and errors to determine the best generation among candidate samples.

7 Limitations & Risks

While our approach performs competitively with most methods that use closed models, the choice to use smaller language models poses an inherent challenge on the capability of generating complex SQL statements. This was observed to be the case when analyzing responses qualitatively based on query complexity. We found that reliance on smaller language models necessitated the use of the critiquing step for adequate performance, which increases the complexity of the inference pipeline, even though the overall computational demand is lower than competing methods. We leave further investigation of the effects of critiquing larger open models, the effects of multi-sample critiquing on them, and the trade-offs around cost and performance to future work.

There can also be risks that arise when relying on text-to-SQL models or more generally agentic workflows involving LLMs. Such models could inadvertently, or by way of an attack, output SQL queries that damage the databases which are accessible, such as by dropping tables. Our proposed method MSc-SQL is designed to automatically execute generated SQL code on the database and further process the outputs. Hence, if the generated code is damaging or malicious it could be automatically executed leading to harms. Any such workflow that automatically runs generated code should be implemented with guardrails to prevent permanent damage, for instance by backing up databases, and segregating the agentic computing environment from other production systems.

References

- Maciej Besta, Nils Blach, Ales Kubicek, Robert Gerstenberger, Michal Podstawski, Lukas Gianinazzi, Joanna Gajda, Tomasz Lehmann, Hubert Niewiadomski, Piotr Nyczyk, and Torsten Hoefler. 2024. Graph of thoughts: Solving elaborate problems with large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17682–17690.
- Bradley Brown, Jordan Juravsky, Ryan Ehrlich, Ronald Clark, Quoc V Le, Christopher Ré, and Azalia Mirhoseini. 2024. Large language monkeys: Scaling inference compute with repeated sampling. *arXiv:2407.21787*.
- Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2024. Teaching Large Language Models to Self-Debug. In *The Twelfth International Conference on Learning Representations*.
- Paul F Christiano, Jan Leike, Tom Brown, Miljan Martić, Shane Legg, and Dario Amodei. 2017. Deep reinforcement learning from human preferences. In *Advances in Neural Information Processing Systems*, volume 30.
- Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, Christopher Hesse, and John Schulman. 2021. Training verifiers to solve math word problems. *arXiv:2110.14168*.
- Tim Dettmers, Artidoro Pagnoni, Ari Holtzman, and Luke Zettlemoyer. 2023. QLoRA: Efficient Fine-tuning of Quantized LLMs. In *Advances in Neural Information Processing Systems*, volume 36, pages 10088–10115.
- Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, Lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot text-to-SQL with ChatGPT. *arXiv:2307.07306*.
- Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Amy Yang, Angela Fan, et al. 2024. The Llama 3 Herd of Models. *arXiv:2407.21783*.
- 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.
- Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. 2021. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*.
- Robert Irvine, Douglas Boubert, Vyas Raina, Adian Liusie, Ziyi Zhu, Vineet Mudupalli, Aliaksei Koshuk, Zongyi Liu, Fritz Cremer, Valentin Assassi, Christie-Carol Beauchamp, Xiaoding Lu, Thomas Rialan, and William Beauchamp. 2023. Rewarding chatbots for real-world engagement with millions of users. *arXiv:2303.06135*.
- Albert Q. Jiang, Alexandre Sablayrolles, Arthur Mensch, Chris Bamford, Devendra Singh Chaplot, Diego de las Casas, Florian Bressand, Gianna Lengyel, Guillaume Lample, Lucile Saulnier, Léo Renard Lavaud, Marie-Anne Lachaux, Pierre Stock, Teven Le Scao, Thibaut Lavril, Thomas Wang, Timothée Lacroix, and William El Sayed. 2023. Mistral 7b. *arXiv:2310.06825*.
- Nathan Lambert, Valentina Pyatkin, Jacob Morrison, LJ Miranda, Bill Yuchen Lin, Khyathi Chandu, Nouha Dziri, Sachin Kumar, Tom Zick, Yejin Choi, Noah A. Smith, and Hannaneh Hajishirzi. 2024. Rewardbench: Evaluating reward models for language modeling. *arXiv:2403.13787*.

- Dongjun Lee, Choongwon Park, Jaehyuk Kim, and Heesoo Park. 2024. MCS-SQL: Leveraging Multiple Prompts and Multiple-Choice Selection For Text-to-SQL Generation. *arXiv:2405.07467*.
- Fei Li and H. V. Jagadish. 2014. [Constructing an interactive natural language interface for relational databases](#). *Proc. VLDB Endow.*, 8(1):73–84.
- Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023a. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 37, pages 13067–13075.
- 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*, 2(3):1–28.
- Jinyang Li, Binyuan Hui, Ge Qu, Jiayi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023b. 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*, volume 36, pages 42330–42357.
- Junyou Li, Qin Zhang, Yangbin Yu, Qiang Fu, and Deheng Ye. 2024b. More agents is all you need. *arXiv:2402.05120*.
- Yifei Li, Zeqi Lin, Shizhuo Zhang, Qiang Fu, Bei Chen, Jian-Guang Lou, and Weizhu Chen. 2022a. Making large language models better reasoners with step-aware verifier. *arXiv:2206.02336*.
- Yujia Li, David Choi, Junyoung Chung, Nate Kushman, Julian Schrittwieser, Rémi Leblond, Tom Eccles, James Keeling, Felix Gimeno, Agustin Dal Lago, et al. 2022b. Competition-level code generation with alphacode. *Science*, 378(6624):1092–1097.
- Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. 2024. Let’s verify step by step. In *The Twelfth International Conference on Learning Representations*.
- Nelson F Liu, Kevin Lin, John Hewitt, Ashwin Paranjape, Michele Bevilacqua, Fabio Petroni, and Percy Liang. 2024. Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, 12:157–173.
- Karime Maamari, Fadhil Abubaker, Daniel Jaroslawicz, and Amine Mhedhbi. 2024. The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models. *arXiv:2408.07702*.
- OpenAI. 2023. [GPT-4 Technical Report](#).
- Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. In *Advances in Neural Information Processing Systems*, volume 35, pages 27730–27744.
- Ana-Maria Popescu, Alex Armanasu, Oren Etzioni, David Ko, and Alexander Yates. 2004. Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In *COLING 2004: Proceedings of the 20th International Conference on Computational Linguistics*, pages 141–147.
- Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157.
- 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*, volume 36.
- Mohammadreza Pourreza and Davood Rafiei. 2024. DTS-SQL: Decomposed Text-to-SQL with Small Large Language Models. *arXiv:2402.01117*.
- Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct Preference Optimization: Your Language Model is Secretly a Reward Model. In *Advances in Neural Information Processing Systems*, volume 36, pages 53728–53741.
- Daking Rai, Bailin Wang, Yilun Zhou, and Ziyu Yao. 2023. Improving generalization in language model-based text-to-SQL semantic parsing: Two simple semantic boundary-based techniques. *arXiv:2305.17378*.
- Machel Reid, Nikolay Savinov, Denis Teplyashin, Dmitry Lepikhin, Timothy Lillicrap, Jean-baptiste Alayrac, Radu Soricut, Angeliki Lazaridou, Orhan Firat, Julian Schrittwieser, et al. 2024. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *arXiv:2403.05530*.
- Morgane Riviere, Shreya Pathak, Pier Giuseppe Sessa, Cassidy Hardin, Surya Bhupatiraju, Léonard Hussenot, Thomas Mesnard, Bobak Shahriari, Alexandre Ramé, et al. 2024. Gemma 2: Improving open language models at a practical size. *arXiv:2408.00118*.
- Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. 2024. Scaling LLM test-time compute optimally can be more effective than scaling model parameters. *arXiv:2408.03314*.

- Weiwei Sun, Lingyong Yan, Xinyu Ma, Shuaiqiang Wang, Pengjie Ren, Zhumin Chen, Dawei Yin, and Zhaochun Ren. 2023. Is ChatGPT good at search? investigating large language models as re-ranking agents. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*.
- Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual harnessing for efficient SQL synthesis. *arXiv:2405.16755*.
- Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.
- Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Qian-Wen Zhang, Zhao Yan, and Zhoujun Li. 2023. MAC-SQL: Multi-agent collaboration for text-to-SQL. *arXiv:2312.11242*.
- Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2022. Self-consistency improves chain of thought reasoning in language models. *arXiv:2203.11171*.
- Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Brian Ichter, Fei Xia, Ed Chi, Quoc V Le, and Denny Zhou. 2022. Chain-of-thought prompting elicits reasoning in large language models. In *Advances in Neural Information Processing Systems*, volume 35.
- Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Tom Griffiths, Yuan Cao, and Karthik Narasimhan. 2023a. Tree of thoughts: Deliberate problem solving with large language models. In *Advances in Neural Information Processing Systems*, volume 36, pages 11809–11822.
- Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. 2023b. **ReAct: Synergizing Reasoning and Acting in Language Models**. In *The Eleventh International Conference on Learning Representations*.
- Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, David Radev, Dragomir Chiang, Julia Hockenmaier, and Jun'ichi Tsujii. 2018. **Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL task**. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 3911–3921.
- Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D Goodman. 2024. Quiet-star: Language models can teach themselves to think before speaking. *arXiv:2403.09629*.
- Di Zhang, Jiatong Li, Xiaoshui Huang, Dongzhan Zhou, Yuqiang Li, and Wanli Ouyang. 2024a. Accessing GPT-4 level Mathematical Olympiad Solutions via Monte Carlo Tree Self-refine with LLaMa-3 8B. *arXiv:2406.07394*.
- Xin Zhang, Yanzhao Zhang, Dingkun Long, Wen Xie, Ziqi Dai, Jialong Tang, Huan Lin, Baosong Yang, Pengjun Xie, Fei Huang, Meishan Zhang, Wenjie Li, and Min Zhang. 2024b. mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval. *arXiv:2407.19669*.
- Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, Hao Zhang, Joseph E Gonzalez, and Ion Stoica. 2023. Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. In *Advances in Neural Information Processing Systems*, volume 36.

A Appendix

We specify example prompts and outputs used to train the schema linking, SQL generation and multi-sample critiquing models.

A.1 Schema Linking

```
User: As an experienced and professional database administrator, your task is to analyze a user question and a database schema to provide relevant information. You are given an SQL Question, "Evidence" which is information that you need to use to solve the question, "DB schema" containing the database schema.
```

```
Identify and list all the relevant tables names from the DB schema based on the user question, database schema and evidence provided. Make sure you include all of them.
```

```
SQL Question: What is the highest eligible free rate for K-12 students in the schools in Alameda County?
```

```
Evidence: Eligible free rate for K-12 = `Free Meal Count (K-12)' / `Enrollment (K-12)'
```

```
DB schema: california_schools
```

```
TABLE "frpm"
```

```
(
```

```
  "CDSCode" TEXT primary key,
```

```
  "Academic Year" TEXT,
```

```
  "County Code" TEXT,
```

```
  "District Code" INTEGER,
```

```
  ...
```

```
  2013-14 CALPADS Fall 1 Certification Status INTEGER,
```

```
  foreign key (CDSCode) references schools(CDSCode)
```

```
)
```

```
TABLE "satscores"
```

```
(
```

```
  "cds" TEXT primary key,
```

```
  "rtype" TEXT,
```

```
  "sname" TEXT,
```

```
  "dname" TEXT,
```

```
  ...
```

```
  foreign key (cds) references schools(CDSCode)
```

```
)
```

```
TABLE "schools"
```

```
(
```

```
  "CDSCode" TEXT primary key,
```

```
  "NCESDist" TEXT,
```

```
  "NCESSchool" TEXT,
```

```
  "StatusType" TEXT,
```

```
  ...
```

```
  LastUpdate DATE,
```



```
)
Assistant:
```json
{
 "tables": ["frpm", "schools"]
}
```
```

A.2 SQL Generation

User: You are given an SQL Question, "Evidence" which is information that you need to use to solve the question, 'DB schema' containing the database schema.

Think step by step and solve the question by coming up with the correct SQL statement that solves the question.

Important things to keep in mind:

1. Only use the tables that are required to solve the task.
2. Use the "evidence" and sample values and column descriptions given to you for reasoning.
3. Don't forget keywords such as DISTINCT, WHERE, GROUP BY, ORDER BY, LIMIT, etc. if needed.

SQL Question: What is the highest eligible free rate for K-12 students in the schools in Alameda County?

Evidence: Eligible free rate for K-12 = `Free Meal Count (K-12)` / `Enrollment (K-12)`

DB schema: california_schools

TABLE "frpm"

```
(
  "CDSCode" TEXT primary key, -- Example: "10621660114355",
  "10621176113575", "10621256113849",
  "Academic Year" TEXT, -- Example: "2014-2015",
  "County Name" TEXT, -- Example: "Alameda", "Contra Costa", "Santa
  Clara",
  "District Name" TEXT, -- Example: "Alameda County Offic...", "San
  Francisco County...", "San Mateo County Off...",
  ...
  foreign key (CDSCode) references schools(CDSCode)
)
```

TABLE "schools"

```
(
  "CDSCode" TEXT primary key, -- Example: "04755070000000",
  "04615490000000", "04746820000000",
  "County" TEXT, -- County name Example: "Alameda", "Contra Costa", "
  Santa Clara",
  "District" TEXT, -- Example: "Alameda County Offic...", "San
  Francisco County...", "San Mateo County Off...",
```

```
"School" TEXT, -- Example: "Alameda County Oppor...", "Alameda
County Speci...", "Alameda City Unified...",
"City" TEXT, -- Example: "Alameda", "Santa Clara", "San Mateo",
...
)
```

Reminder of the SQL Question: What is the highest eligible free rate for K-12 students in the schools in Alameda County?

Assistant:

```
```json
{
 "sql": "SELECT `Free Meal Count (K-12)` / `Enrollment (K-12)`
FROM frpm WHERE `County Name` = 'Alameda' ORDER BY (CAST(`Free
Meal Count (K-12)` AS REAL) / `Enrollment (K-12)`) DESC LIMIT 1"
}
```
```

A.3 Multi-Sample Critiquing

User: You are an SQL, database expert. A previous user was given a task of writing a SQL query given a question prompt. The user wrote 2 possible SQL queries. One is correct and one is wrong given the question. Your task is to use the question and your expertise to decide which one is correct. Here is the question prompt:

You are given an 'SQL Question', 'Evidence' which is information that you need to use to solve the question, 'DB schema' containing the database schema.

Think step by step and solve the question by coming up with the correct SQL statement that solves the question.

Important things to keep in mind:

1. Only use the tables that are required to solve the task.
2. Use the "evidence" and sample values and column descriptions given to you for reasoning.
3. Don't forget keywords such as DISTINCT, WHERE, GROUP BY, ORDER BY, LIMIT, etc. if needed.

SQL Question: Please list the phone numbers of the direct charter-funded schools that are opened after 2000/1/1.

Evidence: Charter schools refers to `Charter School (Y/N)` = 1 in the frpm

DB schema: Database: california_schools

TABLE "frpm"

```
(
  CDSCode TEXT primary key, -- Example: "10621661030675",
  "04755070433953", "10621661035831",
  "Academic Year" TEXT, -- Example: "2014-2015",
  "County Code" TEXT, -- Example: "33", "48", "49",
```

```

"District Code" INTEGER, -- Example: 10017, 31609, 31617,
...
foreign key (CDSCode) references schools(CDSCode)
)

TABLE "schools"
(
CDSCode TEXT primary key, -- Example: "01316090000000",
"04755070000000", "04755070433953",
NCESDist TEXT, -- This field represents the 7-digit National Center
for Educational Statistics (NCES) school
...
LastUpdate DATE, -- Example: 2015-06-23, 2015-09-01, 2015-06-18,
)

```

Reminder of the SQL Question: Please list the phone numbers of the direct charter-funded schools that are opened after 2000/1/1.

The following are the two SQL queries written by the user along with the sample results they generated. One is correct and one is wrong. You need to decide which one is correct.

```

1: SELECT DISTINCT T2.Phone, T1.CDSCode FROM frpm AS T1 INNER JOIN
schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/
N)` = 1 AND T2.OpenDate > '2000-01-01' AND T1.FundingType = 'Directly
funded'

```

Results of 1st SQL:

```

+-----+-----+
| Phone           | CDSCode          |
+-----+-----+
(510) 596-8901	01100170109835
(510) 563-1504	01100170118489
(510) 146-7526	01100170130419
+-----+-----+

```

```

2: SELECT T2.Phone FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.
CDSCode = T2.CDSCode WHERE T1.`Charter Funding Type` = 'Directly
funded' AND T2.OpenDate > '2000-01-01'

```

Results of 2nd SQL:

```

+-----+-----+
| Phone           |
+-----+-----+
| None            |
| (510) 596-8901  |
| None            |
| (510) 686-4131  |
| (510) 452-2063  |
+-----+-----+

```

Provide the number of the correct SQL.

```

Assistant:
```json
{
 "correct_sql": "2"
}
```

```

B Latency and Computational Cost Analysis

To evaluate the trade-off between latency and performance, we measured the inference time of our baseline model f_{gen}^M and compared it with various configurations in Table 2. All models are assumed to be preloaded into memory, and SQL generation is parallelized where no sequential dependencies exist. Our method incurs a nearly $1.7\times$ increase in latency compared to the standalone model, but this results in a performance gain of nearly 10% on the BIRD Dev set, demonstrating an effective trade-off between inference speed and accuracy.

A direct latency comparison with other Text2SQL approaches in Table 1 is infeasible due to proprietary models and the unavailability of open-source implementations. Instead, we estimate the computational cost in FLOPs for the strongest baseline by assuming a fixed context length of 2048 tokens per model invocation. We utilize the `calculate-flops.pytorch`² library to compute total FLOPs across all function calls. The results are summarized in Table 6.

Compared to the CHESS baseline, which employs a 70B LLaMA model with multiple queries, our method requires over 10 times less compute while achieving superior performance. This highlights the efficiency of leveraging additional test-time computation with smaller models to outperform larger models while reducing computational costs.

| Model Family | Params. | Dev EX % | Latency (rel. to baseline) | FLOPs (TFLOPs) |
|---|---------|----------|----------------------------|----------------|
| f_{gen}^M | 7B | 56.4 | 1.00 | 28.59 |
| f_{gen}^L | 8B | 54.1 | 0.89 | 28.59 |
| f_{gen}^G | 9B | 55.0 | 1.10 | 34.09 |
| $f_{\text{link}}^M + f_{\text{gen}}^M$ | 7B | 61.3 | 1.24 | 57.18 |
| $f_{\text{link}}^M + f_{\text{gen}}^L$ | 8B | 60.0 | 1.21 | 57.18 |
| $f_{\text{link}}^M + f_{\text{gen}}^G$ | 9B | 60.8 | 1.28 | 62.68 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^M\} \rightarrow f_{\text{msc}}^M$ | 7B | 64.0 | 1.42 | 114.36 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^L, f_{\text{gen}}^L\} \rightarrow f_{\text{msc}}^M$ | 8B | 63.5 | 1.50 | 114.36 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^G, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$ | 9B | 62.7 | 1.53 | 125.36 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^M, f_{\text{gen}}^M\} \rightarrow f_{\text{msc}}^M$ | 7B | 64.8 | 1.55 | 142.95 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^L, f_{\text{gen}}^L, f_{\text{gen}}^L\} \rightarrow f_{\text{msc}}^M$ | 8B | 64.1 | 1.54 | 142.95 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^G, f_{\text{gen}}^G, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$ | 9B | 62.9 | 1.69 | 159.45 |
| $f_{\text{link}}^M \rightarrow \{f_{\text{gen}}^M, f_{\text{gen}}^L, f_{\text{gen}}^G\} \rightarrow f_{\text{msc}}^M$ | 9B | 65.6 | 1.68 | 148.45 |
| CHESS+LLaMA-3 | 70B | 61.5 | NA | 1682.28 |

Table 6: Comparison of latency and FLOPs across different model configurations on the BIRD dataset.

²<https://github.com/MrYxJ/calculate-flops.pytorch>