# Evaluation of Open-Source Language Models for Text-to-SQL

Omitted for double-blind review

***Abstract*— Text-to-SQL enables both technical and non-technical users to efficiently retrieve information from databases using natural language instead of specific database dialects. While state-of-the-art solutions often rely on large proprietary models, they present challenges related to privacy, cost, and deployment efficiency. In this work, we conducted a systematic review and evaluated open-source language models, focusing on smaller models (up to 14 billion parameters) that can be optimized and deployed on consumer-grade GPUs. Our approach achieved 62.58% execution accuracy on the Bird benchmark, outperforming previous solutions with similar parameter counts. Furthermore, we systematically explored the limitations of smaller models, identifying both their potential and key bottlenecks. These results highlight the feasibility of scaling down while maintaining competitive performance and efficiency.**

***Index Terms*— LLM, Text-to-SQL, NL2SQL**

## I. Introduction

Text-to-SQL is the task of translating a user's natural language question into an equivalent SQL query that retrieves the desired information from a database. With the advent of large language models (LLMs), this problem is typically addressed by embedding the user's question into an LLM prompt, alongside the database schema and supplementary information to better describe the database content. Current literature in this domain is predominantly shaped by solutions that leverage very large proprietary language models with advanced reasoning capabilities. However, these approaches face significant limitations in terms of cost, efficiency, and adaptability. Relying on proprietary language models with restricted access, often guarded by paywalls, greatly increases the cost of deploying such systems. Beyond the high costs of API usage, there are serious concerns about data privacy, as organizations are often hesitant to send sensitive information, such as customer account details, to external servers. This reluctance, particularly from large companies, poses a major barrier to the widespread adoption of such solutions. Recent studies have demonstrated that open-source language models, when effectively prompted and configured, can achieve performance levels comparable to proprietary models. Nevertheless, these solutions often rely on large-scale models with over 15 billion parameters. While such models can be deployed on local servers, their adoption remains prohibitive for small businesses due to the high costs associated with enterprise-grade graphics processing units (GPUs). The expense of acquiring and maintaining the necessary hardware often outweighs the benefits for smaller organizations. Furthermore, real-world text-to-SQL applications require periodic updates to the model parameters to ensure continuous improvements and adaptability to specific customer use cases. Fine-tuning large language models is often impractical, as it demands expensive GPUs with substantial memory capacity. This challenge underscores the need for more cost-efficient, adaptable, and resource-conscious solutions in the text-to-SQL domain.

To address these limitations, we investigated the performance of various open-source large language models, aiming to eliminate the reliance on models with tens of billions of parameters. All selected models contain fewer than 15 billion parameters, enabling fine-tuning on consumer-grade GPUs with at least 24 GB of memory. The reduced hardware requirements of these smaller models mitigate the challenges of previous solutions, offering nearly equivalent performance to very large language models. Moreover, their lower deployment costs make them accessible to a wide range of companies, from small businesses to large enterprises. Running these smaller models on local servers alleviates privacy concerns, as sensitive data no longer needs to leave an organization's network. Additionally, these models can be updated more quickly and efficiently, allowing for continuous improvement and adaptation to novel use cases. We evaluated the execution accuracy of the selected models on the challenging BIRD dataset, comparing our results to prior solutions. The results reveal that smaller language models can deliver competitive performance at significantly lower computational costs. Furthermore, we demonstrated that these models can achieve execution accuracy comparable to large proprietary models when provided with accurate database schema information, highlighting the importance of schema linking as a pre-processing step. Finally, we outlined practical guidelines for fine-tuning smaller models, emphasizing techniques to manage memory allocation during the process.

## II. Related works

The development of simplified interfaces that enable users to query relational databases using natural language has been an active area of research for decades. Early approaches relied on rule-based templates to generate SQL queries from user questions, but demonstrated poor performance when tested in more complex use cases [1]–[3]. The rise of deep learning and natural language processing gave a new momentum to the field, leading to the adoption of recurrent neural networks for translating natural language into SQL queries [4]–[12]. Despite some initial promising results, deep learning techniques demonstrated poor reasoning capabilities when asked to generate more complex queries. In addition, deep learning models

often required complex encoding to represent the database schema and the generation of SQL queries still followed rigid templates to avoid trivial hallucinations [13], [14]. Similarly to what happened in other application domains, large language models completely transformed text-to-SQL as well, quickly overcoming the performance of all previous methods thanks to the impressive reasoning capabilities demonstrated by these models [15]–[17]. Early approaches directly prompted LLMs with the database schema and the user question without applying any fine-tuning to the base version of the model. Then, researchers quickly found that aligning LLMs to the specific text-to-SQL task delivers superior performance with respect to the pre-trained versions [18], [19]. Over the past year, several strategies have emerged to further improve the accuracy of existing LLM-based approaches. Schema linking is a widely adopted technique used to select the tables and columns that are actually useful for generating the SQL query. On the one hand, small language models with up to 7 billion parameters showed some improvement when aided by schema linking [20]. On the other hand, large language models with trillions of parameters seemed to perform better without schema linking thanks to their ability in handling very long input sequences [21]. Recent works also proposed to use sophisticated retrieval techniques to extrapolate relevant keywords from the user question and link them to the most similar values from the database [22]. The extracted values are typically put close to their columns in the database schema description to further help the LLM in the schema linking process. SQL generation is often followed by a query fixer which is in charge of generating new SQL queries in case of execution errors. This step can be executed multiple times to increase the chance of success. In fact, researchers found that generating multiple SQL queries for the same question significantly increase the chance of finding the correct answer among the set of candidates. Then, a selection agent is usually employed to rank the generated candidates and select the most likely correct SQL query for the final execution on the database [23], [24].

Currently, large proprietary language models are the leading solution for text-to-SQL tasks. Proprietary models such as Gemini and GPT-4 outperform smaller alternatives across all existing text-to-SQL benchmarks, significantly surpassing the execution accuracy of more compact models. This success has led many researchers to focus solely on these high-performing proprietary models, often neglecting important considerations related to the costs and efficiency of closed-source solutions. In contrast, our work aims to explore the potential of smaller language models by investigating their true capabilities in text-to-SQL tasks. To achieve this, we focus on identifying promising open-source models that can run efficiently on consumer-grade GPUs, and we propose a lightweight text-to-SQL framework designed to maximize their execution accuracy. In this paper, we address several unresolved research questions in the text-to-SQL space, which form the core of our contributions.

- **Which is the best open-source LLM for text-to-SQL?** We compare multiple open-source language models to identify the best-performing solution currently available. Our focus is on smaller language models with at most 15 billion parameters that can be fine-tuned on consumer-grade GPUs with 24GB of memory.
- **Is schema linking effective in improving the accuracy of small language models?** We investigate the impact of schema linking on the performance of the selected open-source models, exploring its potential as a preprocessing strategy to enhance the performance of smaller language models.
- **What is the optimal configuration for Low Rank Adaptation (LoRA)?** We conduct an in-depth study to determine the best combination of LoRA hyperparameters to maximize model performance while minimizing the memory footprint during the fine-tuning process.
- **What contextual information should be included in the prompt?** We examine the impact of incorporating different types of contextual information in the prompt, aiming to identify which elements enhance execution accuracy without exceeding the model's context length.

## III. Methodology

Inspired by the work of [20], we implemented a simple two-stage framework consisting of *schema linking* and *SQL generation*. For both stages, we fine-tune a single open-source LLM to generate SQL queries based on the following input: i) the full database schema, ii) the user question, and iii) optionally, an evidence. In the schema linking phase, the fine-tuned LLM first generates a SQL query, from which we extract the relevant tables and columns to form a filtered database schema. In the subsequent SQL generation phase, the same LLM is applied with the filtered schema to generate the final SQL query to be executed on the database engine. The complete pipeline of our proposed text-to-SQL framework is shown in Figure 1. Below, we provide additional details on the schema linking strategy implemented in our framework.

### A. Schema linking

Schema linking generally refers to the task of identifying the tables and columns to be used in the SQL query that answers the user's question. Previous studies have shown that schema linking can be beneficial as a preprocessing step for small language models with limited reasoning capabilities [20]. In contrast, schema linking appears to be detrimental when applied to large proprietary language models, as it may remove potentially useful columns from the full database schema [21].

Many prior approaches rely on one or more LLM agents to predict the tables and columns needed for the final SQL query [22]. In this work, we propose a simpler schema linking technique that achieves the same accuracy as previous methods while using only a single LLM call. The core assumption of our approach is that language models are inherently better at generating SQL code than at predicting the correct tables and columns for the query [25]. As a result, our schema linking technique first generates a preliminary SQL query from the user's question, then extracts the relevant tables and columns from the generated query. While the generated SQL query may not be entirely accurate, the extracted tables and columns are

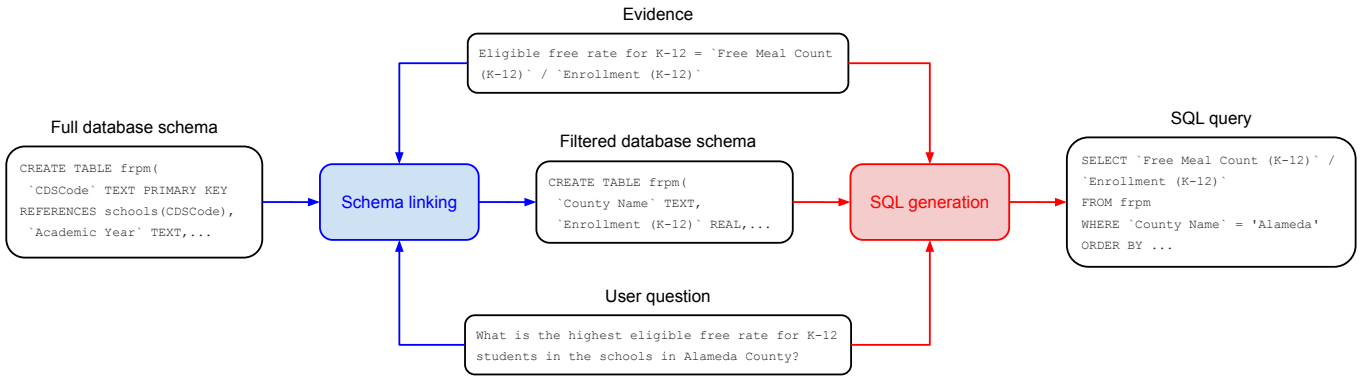


Fig. 1. Overview of the proposed text-to-SQL framework.

often correct and can be used to refine the database schema for improved results in the subsequent SQL generation phase. An example of a generated SQL query with highlighted tables and columns is shown in Figure 2.

To enhance the recall of our schema linking methodology, we expand the initial set of elements extracted from the predicted query by incorporating additional columns. Specifically, we include tables and columns explicitly mentioned in the user's question or the optional evidence. This is achieved by identifying direct references to column names within the question or evidence. Additionally, we incorporate all primary and secondary keys, as we found that they improve recall with only a slight reduction in precision. In summary, our schema linking technique derives predicted columns from four sources: i) the LLM-generated query, ii) the user's question, iii) the provided evidence, and iv) the primary and secondary keys of the database.

```
SELECT COUNT(T1.CDSCode)
FROM frpm AS T1
INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode
WHERE T2.County = 'Los Angeles'
AND T1.`Free Meal Count (K-12)` > 500
AND T1.`FRPM Count (K-12)` < 700
```

Fig. 2. Example of predicted query with the tables (red) and columns (blue) extracted by our schema linking strategy.

### B. Models

The language models evaluated in this paper were selected based on two criteria. First, their parameters must be freely available for download to enable local fine-tuning. Second, the model must have no more than 15 billion parameters to ensure compatibility with consumer-grade GPUs for both fine-tuning and inference. Applying these criteria, we selected 12 open-source language models from the available alternatives. The exact model names, along with their parameter counts and maximum context lengths, are provided in Table I. Context length is particularly important as it determines how many tables and columns can be included in the LLM prompt. All models were fine-tuned for one epoch using a batch size of 32 and a learning rate of 0.0001. Fine-tuning was performed efficiently on a smaller subset of parameters using Low-Rank Adaptation (LoRA) combined with 4-bit quantization. Each training epoch took approximately nine hours. For reproducibility, the specific training arguments used in all fine-tuning experiments are detailed in Table II.

TABLE I
OPEN-SOURCE LLMS EVALUATED IN OUR EXPERIMENTS.

| Model name | Params | Max length |
|---|---|---|
| Llama-3.2-3B-Instruct | 3.21B | 131k |
| Meta-Llama-3.1-8B-Instruct | 8.03B | 131k |
| CodeLlama-7b-Instruct-hf | 6.74B | 16k |
| gemma-2-9b-it | 9.24B | 8k |
| codegemma-7b-it | 8.54B | 8k |
| Qwen2.5-7B-Instruct | 7.62B | 32k |
| Qwen2.5-Coder-7B-Instruct | 7.62B | 32k |
| Qwen2.5-Coder-14B-Instruct | 14.8B | 32k |
| deepseek-coder-7b-instruct | 6.74B | 16k |
| Mistral-7B-Instruct-v0.3 | 7.25B | 32k |
| phi-4 | 14.7B | 16k |
| Phi-3.5-mini-instruct | 3.82B | 131k |

TABLE II
TRAINING HYPERPARAMETERS.

| Parameter | Value |
|---|---|
| batch size | 2 |
| accumulation steps | 16 |
| epochs | 1 |
| learning rate | 0.0001 |
| optimizer | adamw |
| lr scheduler | cosine |
| warmup steps | 5 |
| weight decay | 0.01 |
| max grad norm | 0.3 |
| random seed | 3407 |

### C. Hardware

All experiments were conducted on a server equipped with a single NVIDIA RTX A5000 GPU (24 GB memory). We intentionally used a consumer-grade GPU to demonstrate that our models can operate in memory-constrained environments. Notably, both fine-tuning and inference were successfully performed on the same device without encountering out-of-memory errors. This is a crucial factor in reducing the deployment costs of text-to-SQL solutions in real-world applications.

## IV. Experiments

### A. Dataset and metrics

The following experiments are conducted on the BIRD benchmark [26], a widely recognized dataset consisting of 12,751 distinct pairs of SQL queries and questions spread across 95 databases. BIRD spans 37 application domains, including finance, sports, healthcare, and education, providing a diverse testing ground for model performance across various professional fields. Additionally, BIRD features complex column naming conventions, making the translation task more challenging. Furthermore, its tables often contain numerous columns, increasing prompt length and testing the reasoning capabilities of language models. Table III details the cardinality of the training and development sets used for fine-tuning and evaluation, respectively.

TABLE III
BIRD DATASET.

| Dataset | Samples | #DBs |
|---|---|---|
| Train | 9428 | 71 |
| Dev | 1534 | 11 |

Execution accuracy (EX) is a widely accepted metric for evaluating model performance in text-to-SQL tasks. It is computed by executing both the predicted and ground truth SQL queries on the database and comparing their results. A generated query is considered correct if it produces the same results as the ground truth query, even if the SQL syntax differs. This accounts for the fact that multiple queries can yield the same correct outcome. Execution accuracy is calculated as the number of correct queries divided by the total number of samples in the evaluation set. In summary, it represents the percentage of correctly generated queries in the dataset.

$$EX = \frac{number\ of\ correct\ queries}{total\ number\ of\ queries}$$

### B. Results

*1) Which is the best open-source LLM for text-to-SQL?:* Table IV presents the execution accuracy of the open-source LLMs evaluated in our study. Each model was tested in its base form and after fine-tuning with the full database schema as input. The results reveal several key insights: (1) consistent with prior studies, smaller LLMs show significant improvement in text-to-SQL after fine-tuning for this specific task; (2) LLMs pre-trained for coding tasks consistently outperform general-purpose models not aligned with code generation; and (3) as anticipated, models with larger parameter counts (e.g. 14B) demonstrate higher accuracy compared to smaller models. Among the tested open-source LLMs, QwenCoder-14B emerged as the best-performing model for text-to-SQL.

TABLE IV
EVALUATION OF THE SELECTED OPEN-SOURCE LLMS IN THEIR BASE VERSION AND AFTER FINE-TUNING.

| Model | $EX_{base}$ | $EX_{fine\text{-}tuned}$ |
|---|---|---|
| Llama-3.2-3B-Instruct | 25.29 | 46.28 |
| Meta-Llama-3.1-8B-Instruct | 38.66 | 53.32 |
| CodeLlama-7b-Instruct-hf | 21.32 | 51.50 |
| gemma-2-9b-it | 9.26 | 52.74 |
| codegemma-7b-it | 4.04 | 52.93 |
| Qwen2.5-7B-Instruct | 35.59 | 37.87 |
| Qwen2.5-Coder-7B-Instruct | 49.48 | 52.67 |
| **Qwen2.5-Coder-14B-Instruct** | **55.58** | **59.71** |
| deepseek-coder-7b-instruct | 41.72 | 56.06 |
| Mistral-7B-Instruct-v0.3 | 26.57 | 52.35 |
| phi-4 | 46.02 | 59.39 |
| Phi-3.5-mini-instruct | 27.84 | 46.87 |

*2) Is schema linking effective in improving the accuracy of small language models?:* It is known that schema linking can improve the full schema only if it is actually able to retrieve all the columns needed to generate the correct SQL query. To evaluate the effectiveness of the proposed schema linking method in identifying the correct columns, we computed the average recall and precision in all samples from the BIRD development set. Despite the simplicity of our approach, Table V shows that our methodology is comparable to more advanced schema linking techniques implemented in other works. In particular, we achieved the highest average precision and almost the same recall of the most accurate methods reported in the literature. Apart from that, the most important achievement of our schema linking technique is the significant cost reduction at inference time. In fact, we could get almost the same results of previous works employing multiple agents by using only a single LLM call. In addition, previous methods used multiple retrieval agents combined with sophisticated search algorithms to extract relevant keywords from the user's question and link them to their columns [22]–[24]. In fact, adding relevant keywords beside column names in the schema representation can significantly help the LLM in finding the correct columns that will be used in the SQL query. However, here we decided to avoid implementing these advanced search strategies because they usually require a full indexed copy of the database content, which is highly impractical in a real-world scenario with petabytes of database content. Nevertheless, value retrieval remains an effective strategy when dealing with small databases, which should be considered on the base of the specific use case.

As shown in Table VI, the proposed schema linking technique significantly enhances the performance of nearly all models compared to the full schema scenario reported in Table IV. In particular, schema linking improved the

TABLE V
SCHEMA LINKING RECALL AND PRECISION.

| Method | Recall | Precision |
|---|---|---|
| CHESS | 0.94 | 0.71 |
| XiYan-SQL | 0.95 | 0.75 |
| Our | 0.94 | 0.76 |

execution accuracy of QwenCoder-14B from 59.71 to 62.58, setting a new record among small language models. Table VI also illustrates that these models perform substantially better in an idealized perfect schema scenario, where only the necessary tables and columns for SQL query generation are included. In this setting, QwenCoder-14B achieves a peak execution accuracy of 69.23, approaching the performance of proprietary language models on the same benchmark. Overall, these findings underscore the effectiveness of schema linking for small language models. Moreover, the results from the perfect schema scenario highlight the potential for further improvements through more precise schema linking strategies.

TABLE VI
EVALUATION OF THE SELECTED OPEN-SOURCE LLMS WITH SCHEMA LINKING AND PERFECT SCHEMA SCENARIO.

| Model | $EX_{schema\ linking}$ | $EX_{perfect}$ |
|---|---|---|
| Llama-3.2-3B-Instruct | 48.89 (+2.61) | 62.13 |
| Meta-Llama-3.1-8B-Instruct | 59.26 (+5.94) | 66.10 |
| CodeLlama-7b-Instruct-hf | 54.76 (+3.26) | 63.95 |
| gemma-2-9b-it | 50.59 (-2.15) | 66.43 |
| codegemma-7b-it | 58.67 (+5.74) | 65.51 |
| Qwen2.5-7B-Instruct | 45.96 (+8.09) | 63.36 |
| Qwen2.5-Coder-7B-Instruct | 55.8 (+3.13) | 65.19 |
| **Qwen2.5-Coder-14B-Instruct** | **62.58 (+2.87)** | **69.23** |
| deepseek-coder-7b-instruct | 57.63 (+1.56) | 67.01 |
| Mistral-7B-Instruct-v0.3 | 58.02 (+5.67) | 64.15 |
| phi-4 | 61.80 (+2.41) | 68.58 |
| Phi-3.5-mini-instruct | 47.65 (+0.78) | 63.10 |

*3) Which contextual information should be used in the prompt?:* The information included in the prompt plays a fundamental role in the final performance of the SQL generation model. In addition to the user question and an optional evidence, the LLM is always prompted with the database schema enriched with further contextual information. Here, we investigate the effectiveness of adding different contextual information to the prompt. Based on past literature, we studied the effect of two types of additional information: i) column values and ii) column descriptions. In more detail, the column values report the first three distinct values retrieved from the database and provide examples of the column content. The column description instead provides an extended explanation of the column name and content. Table 3 gives an example of the database schema representation used in our experiments augmented with column values (in blue) and column descriptions (in red). It is also important to notice that every additional information increments the prompt length, potentially exceeding the maximum context length of the model. To measure this effect, we report in Table VII the average and maximum prompt length obtained when adding the different contextual information in the case of full schema representation. In all cases, the prompt length never exceeds the maximum context length of the selected models.

```
CREATE TABLE schools(
 `StatusType` TEXT, -- Description: This field identifies the
status of the district. Examples: ['Closed', 'Active']
 `County` TEXT, -- Description: County name. Examples:
['Alameda'] …
```

Fig. 3. Example of plain schema representation with descriptions (red) and column values (blue).

TABLE VII
AVERAGE AND MAXIMUM PROMPT LENGTH WITH DIFFERENT CONTEXTUAL INFORMATION. THE REPORTED NUMBER OF TOKENS IS COMPUTED WITH THE QWEMCODER-14B TOKENIZER.

| Contextual information | Average | Maximum |
|---|---|---|
| Plain schema | 716 | 2736 |
| Column values | 1817 | 6225 |
| Column descriptions | 2494 | 8099 |

Table VIII shows the execution accuracy of QwenCoder-14B on the BIRD development set when different contextual information is added to the plain database schema. On the one hand, the results reveal that adding database values beside the columns can significantly increase LLM performance, increasing execution accuracy from 56.13 to 59.71. On the other hand, the improvement of column descriptions is less notable, with only a slight increase compared to the case with column values. Overall, both contextual information show a positive effect of the final execution accuracy, motivating their use in text-to-SQL prompts.

TABLE VIII
EXECUTION ACCURACY WHEN DIFFERENT CONTEXTUAL INFORMATION ARE ADDED TO THE PROMPT.

| Contextual information | EX |
|---|---|
| Plain schema | 56.13 |
| Column values | 59.71 (+3.58) |
| Column descriptions | 59.91 (+0.20) |

*4) Which is the best configuration for Low Rank Adaptation?:* Low-Rank Adaptation (LoRA) has been widely used for fine-tuning large language models in text-to-SQL tasks [22]. However, the impact of different rank and alpha parameters in LoRA has not been thoroughly investigated. Optimizing these parameters is crucial for maximizing fine-tuning efficiency while minimizing computational costs. Lower rank values are generally preferred to reduce memory usage during fine-tuning. To identify the optimal rank and alpha pair, we conducted

a grid search aimed at maximizing the execution accuracy of QwenCoder-14B on the BIRD development set. Given the computational cost of full-schema representation, we performed our experiments using a reduced database schema containing only the necessary columns for SQL query generation. As shown in Figure 4, our results indicate that higher alpha values consistently improve execution accuracy, regardless of the chosen rank. Conversely, variations in rank have minimal impact on performance, suggesting that lower rank values should be preferred to optimize memory usage. To illustrate this, Table IX reports the peak memory allocation during fine-tuning with different rank values while keeping alpha constant. Additionally, we observed that including the model head in LoRA's target modules does not yield significant accuracy improvements. Moreover, this approach is often impractical for models with large vocabulary sizes (e.g., exceeding 32k tokens). Similarly, LoRA dropout was found to have negligible benefits and can be omitted. Based on these findings, we present in Table X the optimal LoRA configuration used in all our fine-tuning experiments.

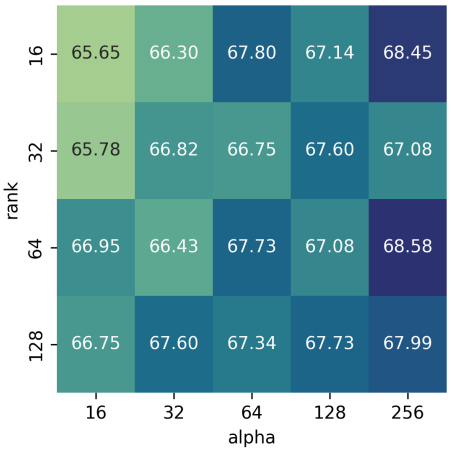


Fig. 4. Execution accuracy with different rank and alpha parameters.

TABLE IX
GPU MEMORY ALLOCATION OF A SINGLE TRAINING STEP WITH QUANTIZED QWENCODER-14B USING DIFFERENT RANKS AND FIXED ALPHA OF 256. WE USED A REAL BATCH SIZE OF 2 SAMPLES WITH 16 GRADIENT ACCUMULATION STEPS.

| Rank | Parameters | Memory |
|---|---|---|
| 16 | 68M | 13.02 GB |
| 32 | 137M | 13.97 GB |
| 64 | 275M | 14.58 GB |
| 128 | 550M | 19.74 GB |

### C. Comparison with other methods

Table XI presents the execution accuracy of our methodology compared to previous approaches evaluated on the BIRD development set. We report results only for methods with publicly available research papers. Additionally, we include the number of parameters for the largest LLM used in each methodology. For proprietary models with undisclosed sizes, such as Gemini and GPT-4, we use the acronym "UNK" to indicate that the parameter count is unknown. As shown in Table XI, our methodology, leveraging QwenCoder-14B for SQL generation and schema linking, surpasses all previous methods that use open-source LLMs with up to 15B parameters. Furthermore, it also outperforms older solutions that incorporated proprietary models like GPT-4 in their frameworks. While our approach does not yet surpass state-of-the-art methods relying on trillion-parameter LLMs, it achieves a strong balance between model size and execution accuracy, as highlighted in Figure 5.

TABLE X
OPTIMAL LORA CONFIGURATION.

| Parameter | Value |
|---|---|
| rank | 16 |
| alpha | 256 |
| modules | q proj, v proj, k proj, o proj, gate proj, up proj, down proj |
| dropout | 0 |
| bias | none |

Overall, our results confirm that the number of parameters is the primary factor determining the performance of a text-to-SQL system. This trend is evident in our comparison of open-source models, where LLMs with 14 billion parameters consistently outperformed smaller models. Similarly, our broader analysis of existing methods highlights that trillion-parameter LLMs achieve significantly higher accuracy than their smaller counterparts. However, we also demonstrate that small language models can unlock their hidden potential when provided with only the necessary columns from the database schema. This finding suggests that enhancing schema linking could further improve execution accuracy, potentially enabling small models to reach proprietary solutions.

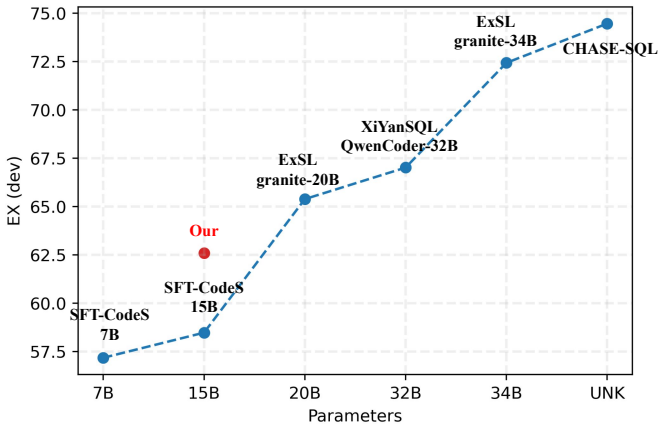


Fig. 5. Comparison of proposed solution (red dot) with available methods in the BIRD leader board based on the execution accuracy obtained on the dev set.

TABLE XI
COMPARISON WITH PREVIOUS METHODS FROM THE BIRD LEADER BOARD BASED ON THE EXECUTION ACCURACY ON THE DEV SET.

| Method | Parameters | EX(dev) |
|---|---|---|
| DIN-SQL | UNK | 50.72 |
| DAIL-SQL | UNK | 54.76 |
| DTS-SQL | 7B | 55.80 |
| TA-SQL | UNK | 56.19 |
| SFT CodeS-7B | 7B | 57.17 |
| MAC-SQL | UNK | 57.56 |
| SFT CodeS-15B | 15B | 58.47 |
| SuperSQL | UNK | 58.50 |
| E-SQL | UNK | 61.60 |
| **Our** | **14B** | **62.58** |
| MCS-SQL | UNK | 63.36 |
| RSL-SQL | UNK | 63.56 |
| E-SQL | UNK | 65.58 |
| XiYan-SQL | 33B | 67.01 |
| Distillery AI | UNK | 67.21 |
| CHESS | UNK | 68.31 |
| XiYan-SQL | UNK | 73.34 |
| CHASE-SQL | UNK | 74.46 |

## V. CONCLUSION

In conclusion, this study highlights the potential of small open-source language models as effective alternatives for text-to-SQL tasks. By systematically evaluating models with up to 14 billion parameters, we demonstrated that these models can achieve competitive execution accuracy while significantly lowering computational costs and addressing key privacy concerns associated with proprietary large language models. Our experiments underscore the importance of schema linking as a crucial pre-processing step, enabling smaller models to better understand database structures and improve query generation accuracy. Additionally, we explored the impact of contextual information in prompts, showing that including column values and descriptions further enhances model performance. Our fine-tuning experiments using Low Rank Adaptation (LoRA) also revealed optimal configurations that maximize efficiency while minimizing memory usage, making these models more accessible for fine-tuning on consumer-grade GPUs. While proprietary models with trillions of parameters still lead in execution accuracy, our findings suggest that with further improvements in schema linking, smaller models can serve as more cost-efficient and adaptable alternatives. Future research should focus on refining schema linking techniques, optimizing prompt engineering strategies, and exploring lightweight fine-tuning methods to further close the performance gap between small and large models while maintaining scalability and privacy benefits.

## REFERENCES


[1] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch, "Natural language interfaces to databases–an introduction," *Natural language engineering*, vol. 1, no. 1, pp. 29–81, 1995.

[2] J. M. Zelle and R. J. Mooney, "Learning to parse database queries using inductive logic programming," in *Proceedings of the national conference on artificial intelligence*, 1996, pp. 1050–1055.

[3] F. Li and H. V. Jagadish, "Constructing an interactive natural language interface for relational databases," *Proceedings of the VLDB Endowment*, vol. 8, no. 1, pp. 73–84, 2014.

[4] B. Wang, R. Shin, X. Liu, O. Polozov, and M. Richardson, "Rat-sql: Relation-aware schema encoding and linking for text-to-sql parsers," *arXiv preprint arXiv:1911.04942*, 2019.

[5] D. Choi, M. C. Shin, E. Kim, and D. R. Shin, "Ryansql: Recursively applying sketch-based slot fillings for complex text-to-sql in cross-domain databases," *Computational Linguistics*, vol. 47, no. 2, pp. 309–332, 2021.

[6] R. Cai, J. Yuan, B. Xu, and Z. Hao, "Sadga: Structure-aware dual graph aggregation network for text-to-sql," *Advances in Neural Information Processing Systems*, vol. 34, pp. 7664–7676, 2021.

[7] R. Cao, L. Chen, Z. Chen, Y. Zhao, S. Zhu, and K. Yu, "Lgesql: line graph enhanced text-to-sql model with mixed local and non-local relations," *arXiv preprint arXiv:2106.01093*, 2021.

[8] G. Katsogiannis-Meimarakis and G. Koutrika, "A survey on deep learning approaches for text-to-sql," *The VLDB Journal*, vol. 32, no. 4, pp. 905–936, 2023.

[9] J. Li, B. Hui, R. Cheng, B. Qin, C. Ma, N. Huo, F. Huang, W. Du, L. Si, and Y. Li, "Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-sql parsing," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 13 076–13 084.

[10] Y. Gan, X. Chen, Q. Huang, M. Purver, J. R. Woodward, J. Xie, and P. Huang, "Towards robustness of text-to-sql models against synonym substitution," *arXiv preprint arXiv:2106.01065*, 2021.

[11] J. Guo, Z. Zhan, Y. Gao, Y. Xiao, J.-G. Lou, T. Liu, and D. Zhang, "Towards complex text-to-sql in cross-domain database with intermediate representation," *arXiv preprint arXiv:1905.08205*, 2019.

[12] J. Qi, J. Tang, Z. He, X. Wan, Y. Cheng, C. Zhou, X. Wang, Q. Zhang, and Z. Lin, "Rasat: Integrating relational structures into pretrained seq2seq model for text-to-sql," *arXiv preprint arXiv:2205.06983*, 2022.

[13] H. Li, J. Zhang, C. Li, and H. Chen, "Resdsql: Decoupling schema linking and skeleton parsing for text-to-sql," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, 2023, pp. 13 067–13 075.

[14] D. Rai, B. Wang, Y. Zhou, and Z. Yao, "Improving generalization in language model-based text-to-sql semantic parsing: Two simple semantic boundary-based techniques," *arXiv preprint arXiv:2305.17378*, 2023.

[15] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. D. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman *et al.*, "Evaluating large language models trained on code," *arXiv preprint arXiv:2107.03374*, 2021.

[16] T. Brown, B. Mann, N. Ryder, M. Subbiah, J. D. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *Advances in neural information processing systems*, vol. 33, pp. 1877–1901, 2020.

[17] L. Ouyang, J. Wu, X. Jiang, D. Almeida, C. Wainwright, P. Mishkin, C. Zhang, S. Agarwal, K. Slama, A. Ray *et al.*, "Training language models to follow instructions with human feedback," *Advances in neural information processing systems*, vol. 35, pp. 27 730–27 744, 2022.

[18] D. Gao, H. Wang, Y. Li, X. Sun, Y. Qian, B. Ding, and J. Zhou, "Text-to-sql empowered by large language models: A benchmark evaluation," *arXiv preprint arXiv:2308.15363*, 2023.

[19] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang, "The dawn of natural language to sql: Are we fully ready?" *arXiv preprint arXiv:2406.01265*, 2024.

[20] M. Pourreza and D. Rafiei, "Dts-sql: Decomposed text-to-sql with small large language models," *arXiv preprint arXiv:2402.01117*, 2024.

[21] K. Maamari, F. Abubaker, D. Jaroslawicz, and A. Mhedhbi, "The death of schema linking? text-to-sql in the age of well-reasoned language models," *arXiv preprint arXiv:2408.07702*, 2024.

[22] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi, "Chess: Contextual harnessing for efficient sql synthesis," *arXiv preprint arXiv:2405.16755*, 2024.

[23] M. Pourreza, H. Li, R. Sun, Y. Chung, S. Talaei, G. T. Kakkar, Y. Gan, A. Saberi, F. Ozcan, and S. O. Arik, "Chase-sql: Multi-path reasoning and preference optimized candidate selection in text-to-sql," *arXiv preprint arXiv:2410.01943*, 2024.

[24] Y. Gao, Y. Liu, X. Li, X. Shi, Y. Zhu, Y. Wang, S. Li, W. Li, Y. Hong, Z. Luo *et al.*, "Xiyan-sql: A multi-generator ensemble framework for text-to-sql," *arXiv preprint arXiv:2411.08599*, 2024.

[25] G. Qu, J. Li, B. Li, B. Qin, N. Huo, C. Ma, and R. Cheng, "Before generation, align it! a novel and effective strategy for mitigating hallucinations in text-to-sql generation," *arXiv preprint arXiv:2405.15307*, 2024.

[26] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Geng, N. Huo *et al.*, "Can llm already serve as a database interface? a big

bench for large-scale database grounded text-to-sqls," *Advances in Neural Information Processing Systems*, vol. 36, 2024.