Blar-SQL: Cheaper, smaller, faster, stronger NL2SQL

JOSÉ MANUEL DOMÍNGUEZ, Blar Spa, Chile
BENJAMÍN ERRAZURIZ, Blar Spa, Chile

Large Language Models (LLMs) have gained considerable notoriety in the field of natural language to SQL tasks (NL2SQL). In this study, we show how task decomposition can greatly benefit LLMs in database understanding and query generation in order to answer human questions with an SQL query.

We fined-tuned open source models, specifically Llama-2 and Code Llama, we combined 2 different models created for 2 different tasks in order to leverage each model understanding to further increase the accuracy of the final SQL query.

We propose a new framework to divide the schema into chunks in order to fit more information into a limited context. Our results are comparable with those obtained by GPT-4 at the same time being ~135 times smaller, 90 times faster and more than 100 times cheaper than GPT-4.

Additional Key Words and Phrases: NL2SQL, LLM, AI, Prompt Decomposition

ACM Reference Format:
José Manuel Domínguez and Benjamín Errazuriz. 2023. Blar-SQL: Cheaper, smaller, faster, stronger NL2SQL. *ACM Trans. Graph.* 37, 4, Article 111 (August 2023), 6 pages.

1 INTRODUCTION

With recent advances in LLMs such as the Open AI's GPT-4 and Meta's Llama-2, the field of translating human language questions into SQL queries, commonly referred to as NL2SQL, has witnessed significant advancements. These state-of-the-art language models have opened up exciting opportunities for improving the accuracy and efficiency of NL2SQL systems, enabling the end user to interact with databases without having a deep understanding of SQL language or the Database's Schema.

Currently, most of the SOTA approaches use LLMs to achieve this goal, particularly GPT-4, which has gained most of the attention as being the largest and most capable model as of this paper's writing. In this paper, we explore how open-source models such as Llama-2 and CodeLlama when fine-tuned can achieve similar or even better results than the ones achieved by GPT-4. Moreover, we explore how dividing complex tasks into smaller steps can significantly benefit these models. We build on DIN-SQL's approach [Pourreza and Rafiei 2023] and fine-tune two custom models, each specialized in completing one sub-task achieving results comparable with GPT-4 on the BIRD-SQL data set.

Another challenge frequently encountered in the text-to-SQL domain is managing context. Large data schemas often exceed the 32K token context limit of the GPT-4 model. To address this, we have introduced a novel method for selecting the necessary schema link to answer a given query. We partition the schema into several prompts, each tailored to fit within the maximum context length. This approach enables models with a context limit of 4K tokens or less to efficiently process databases with extensive schemas and descriptions.

The contributions of this study are summarized:

(1) We divide the task of generating SQL queries into 2 steps, the first is choosing the appropriate columns based on the database schema, column descriptions and other external knowledge that may be required.
(2) A new framework of sub-dividing very large schemas and descriptions into chunks in order to better manage prompt context and include the greatest amount of information possible.
(3) A new idea of using 2 different models trained with different purposes in order to leverage their knowledge and create better results.

2 RELATED WORK

Decomposing complex text-to-SQL tasks has shown great potential, improving the performance of few shots LLMs on the Spider dataset by over 10% [Pourreza and Rafiei 2023]. The main idea behind this is that LLMs respond better when only one task is asked, to achieve this you break the problem into simple pieces and solve each part using prompting, trained models, or symbolic functions [Khot et al. 2022].

SDSQL model introduces the notion of capturing the interactions between the question and the data schema. It uses a series of dependency labels that link question tokens with schema tokens (*W-Col*, *S-Agg*, *W-Op*).

There are several datasets designed for text-to-SQL research, such as Spider [Yu et al. 2018] and WikiSQL [Zhong et al. 2017]. These datasets not only provide a benchmark for comparing various approaches, but they also supply the data necessary for fine-tuning models and achieving significant improvements. In this realm, the BIRD dataset stands out. It encompasses a 33.4GB database emphasizing real-world applications. This benchmark offers a detailed insight into how our frameworks would perform with actual databases, making it the most rigorous benchmark available [Li et al. 2023].

3 METHODOLOGY

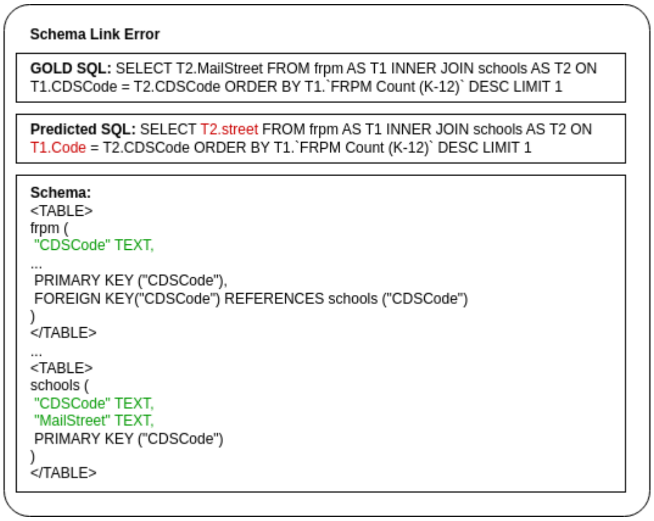
3.1 Error analysis

In order to further understand the current capabilities of open-source LLMs we first evaluated the vanilla version of code-llama against the development dataset of both Spider and Bird. We later took a random subset of 60 data points and classified the errors in order to understand what should be improved.

Authors' addresses: José Manuel Domínguez, jose@getblar.com, Blar Spa, , Santiago, , Chile, ; Benjamín Errazuriz, benjamin@getblar.com, Blar Spa, , Santiago, , Chile,

0730-0301/2023/8-ART111 $15.00
https://doi.org/

The most common error we found where schema linking, this occurred when the model changed the name of some columns or hallucinated columns and table-column associations. The second most common error was when the model failed to understand the content of the database, this error occurred when the model couldn't comprehend the column description and values. Figures 1 and 2 show examples of both errors.



Schema Link Error

GOLD SQL: SELECT T2.MailStreet FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode ORDER BY T1.`FRPM Count (K-12)` DESC LIMIT 1

Predicted SQL: SELECT T2.street FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.Code = T2.CDSCode ORDER BY T1.`FRPM Count (K-12)` DESC LIMIT 1

Schema:

```
<TABLE>
frpm (
 "CDSCode" TEXT,
...
 PRIMARY KEY ("CDSCode"),
 FOREIGN KEY("CDSCode") REFERENCES schools ("CDSCode")
)
</TABLE>
...
<TABLE>
schools (
 "CDSCode" TEXT,
 "MailStreet" TEXT,
 PRIMARY KEY ("CDSCode")
)
</TABLE>
```

Fig. 1. Example of a Schema Link Error
In this example the model got wrong the name of the columns 'CDSCODE' and 'MailStreet'

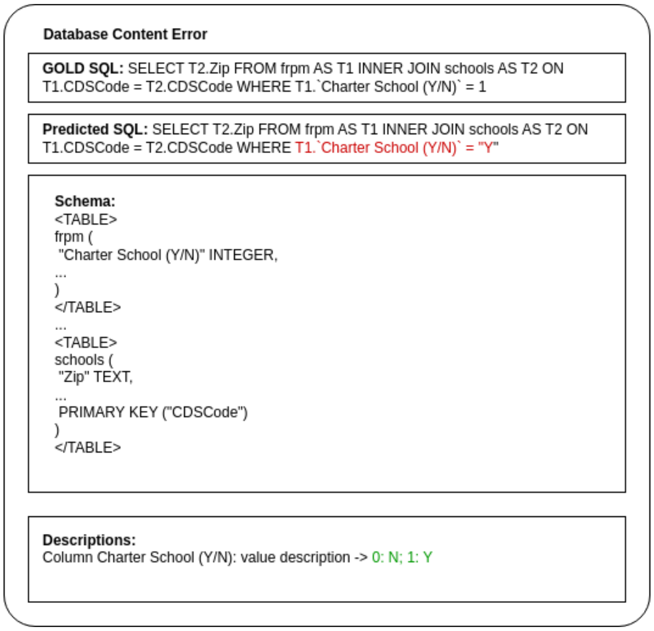
Our results coincide with the results obtained by the team that developed the Bird data set [Li et al. 2023]. The two most common mistakes were "Wrong Schema Linking" (41.6%) and "Misunderstanding Database Content" (40.8%), both occurred because the LLM couldn't understand the database schema correctly or recall the correct structure of the database.

In order to further improve LLMs capability we divided the process into two steps: 1) inferring the schema links (Schema-Link model) and 2) Constructing the SQL using the previously generated schema links (SQL model). The overall architecture is shown in figure 3. The idea is to train both models specifically for their task because it has been proven that dividing a problem into smaller tasks generates better results when it comes to LLM inference [Khot et al. 2022].

3.2 Making the data set

As we mentioned before we used the Bird dataset because is the first benchmark focused on big databases with real applications. Thanks to the work done by the Bird team, you can access 95 databases with 12.751 questions with their respective answer [Li et al. 2023].

In order to create the schema link dataset we reverse-engineered the SQL answers and the schema. This process consisted of extracting only the tables and columns used in the query answers as well as the foreign keys. Figure 4 shows the process of schema link generation. In order to achieve this we used a combination of SQL metadata capturing packages (*sql-metadata* and *sql-parse*) for Python combined with custom-made Regex.



Database Content Error

GOLD SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1

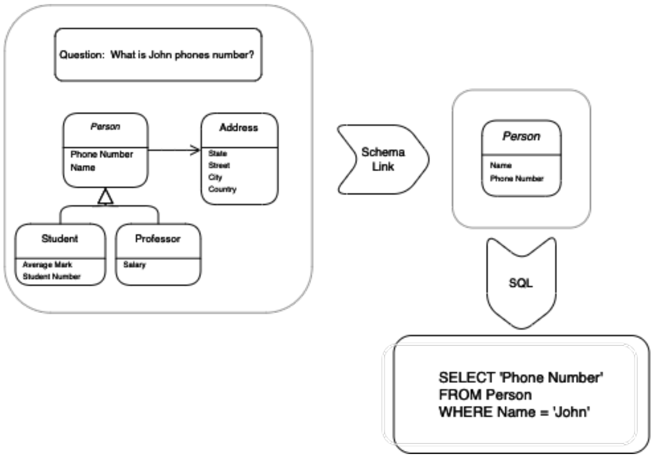
Predicted SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = "Y"

Schema:

```
<TABLE>
frpm (
 "Charter School (Y/N)" INTEGER,
...
)
</TABLE>
...
<TABLE>
schools (
 "Zip" TEXT,
...
 PRIMARY KEY ("CDSCode")
)
</TABLE>
```

Descriptions:
Column Charter School (Y/N): value description -> 0: N; 1: Y

Fig. 2. Example of a database content error
In this example the model assumed that the column 'Charter School (Y/N)' had values "Y" or "N", but in reality the values are 1 or 0

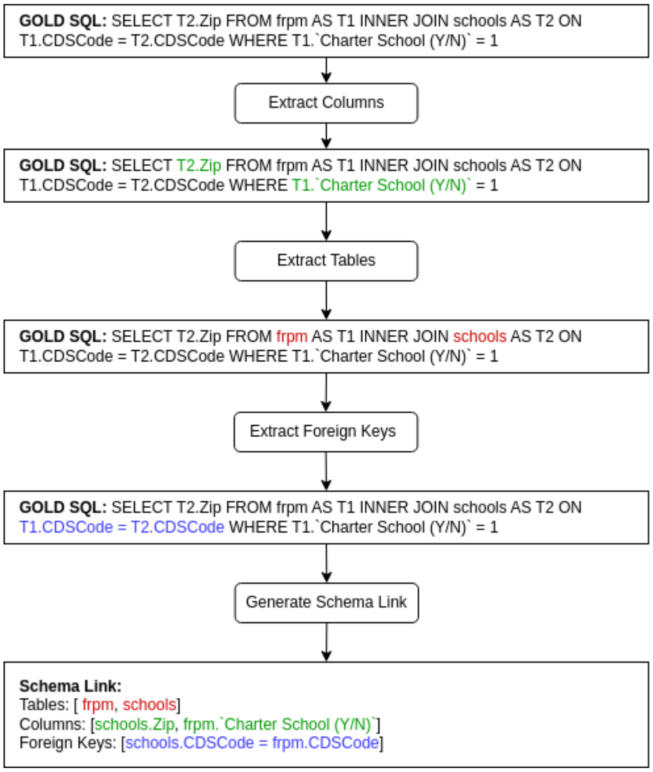


```
SELECT 'Phone Number'
FROM Person
WHERE Name = 'John'
```

Fig. 3. Prediction solution architecture
Here you can see in the first box the question and the database schema, which the Schema-Link model take and infer the tables and columns needed to answer the question. With the schema link the SQL model predicts the final SQL to answer the question.

3.3 Hypothesis

Our solution consists of two models working in two separate tasks, making the schema linking and generating the SQL query. The way that we train each model can impact how these two models interact



```
GOLD SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON
T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1

GOLD SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON
T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1

GOLD SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON
T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1

GOLD SQL: SELECT T2.Zip FROM frpm AS T1 INNER JOIN schools AS T2 ON
T1.CDSCode = T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1

Schema Link:
Tables: [ frpm, schools]
Columns: [schools.Zip, frpm.`Charter School (Y/N)`]
Foreign Keys: [schools.CDSCode = frpm.CDSCode]
```

Fig. 4. Schema Link Generation Process

with each other to come up with a solution. Because of this, we created two hypotheses:

1. A non-trusting interaction: where the SQL model does not rely only on the Schema-Link output, it also has access to the original schema and column descriptions. The schema link are presented as a suggestion more than a fact. By ensuring that the SQL model does not solely rely on the output of the first, there's an added opportunity to correct potential misinterpretations of the schema.
2. A trusting interaction where the SQL model only takes the output of the Schema-Link model to construct the query. This enables us to manage the context length as well as teach the model to focus purely on the construction of the SQL query and the logic behind it.

Furthermore, we hypothesized that using a Llama-2 model for the schema linking part would be beneficial as it has a better world understanding [Rozière et al. 2023] and using code-llama for the SQL part, as it has a better programming background [Touvron et al. 2023]. Unfortunately, due to a lack of time and resources we couldn't compare how a pure llama-2 and pure code-llama solution would perform, so we leave this as a future work.

3.4 Context Management

One big limitation that open-source solutions such as llama-2, code-llama or others have, is the smaller context they have in comparison with Open AI's GPT models. Even GPT-4 32K context may fall short when the database is big enough or if it has detailed column descriptions.

We experimented with two approaches in order to tackle this problem. The first one consisted of reducing the prompt context by excluding certain information such as column descriptions if it exceeded the context. This traditional approach has the limitation of the volume of context that we can provide our models to make a prediction. We call this the non-descriptive approach.

The second approach is a more novel one, in which we propose a new method of dividing the schema linking step according to the maximum context length available.

First, we started by creating the template of the prompt which consisted of four parts. The question, the database schema, column descriptions and finally a hint or external knowledge required in order to answer the question.

Then, we iterated over each table's columns and descriptions, adding one table at a time until the prompt could not further fit another table. The question and external knowledge were fixed in order to always be included. We then divided each question using this method in one or more prompts.

Finally, we extracted the schema links from each segmented prompt, producing a list of schema link predictions. When concatenated, this list provided the essential information required to answer the question. We call this the schema chunking approach and Figure 5 explains the whole process.

3.5 Training process

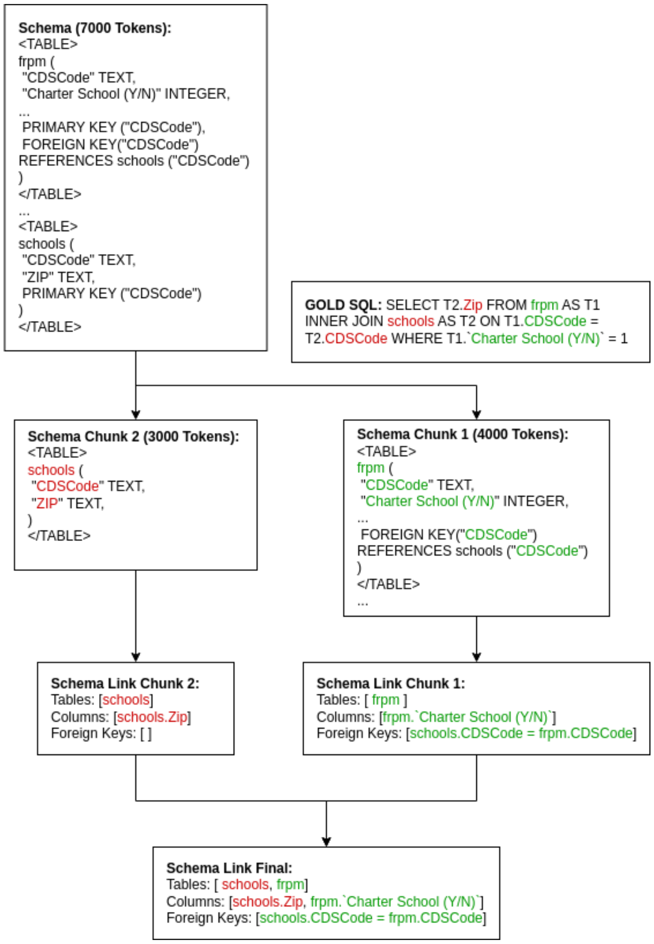
To train the models we used Google Colab's A100 40GB GPU. To achieve this we used 4bit model quantization as well as QLoRA for fine-tuning. The training itself used a combination of the packages available in huggingface such as *transformers*, *peft* and *trl*. The training process usually lasted ~10hrs costing about $20 USD per model.

We did supervised fine-tuning only updating weights from the response (Schema links and SQL queries) similar to the approach taken by Dail-SQL [Gao et al. 2023]. For the dataset, we used BIRD's Training set consisting of 8.952 examples, the dataset was subsequently partitioned randomly, resulting in 7,609 examples left for training, while 1,343 were set aside for validation purposes.

We conducted the training process for a maximum of two epochs, evaluating the model every 500 steps and selecting the best models based on the validation set.

3.5.1 Schema Link models.

- Non-descriptive: In this approach, there was one single prompt for each question, to achieve this we removed the column descriptions if they couldn't be fitted in the context due to length limitations. The model's task was to predict the whole schema link generated previously. Using this approach some examples of the training set were omitted as the pure schema wouldn't fit in the context.
- Chunked: In this approach, each question had N prompts, each with a maximum length of 4096, as we were using the Llama-2 models. Both the schema and descriptions were included in each prompt preventing information loss. The



```
Schema (7000 Tokens):
<TABLE>
frpm (
 "CDSCode" TEXT,
 "Charter School (Y/N)" INTEGER,
...
 PRIMARY KEY ("CDSCode"),
 FOREIGN KEY("CDSCode")
REFERENCES schools ("CDSCode")
)
</TABLE>
...
<TABLE>
schools (
 "CDSCode" TEXT,
 "ZIP" TEXT,
 PRIMARY KEY ("CDSCode")
)
</TABLE>
```

```
GOLD SQL: SELECT T2.Zip FROM frpm AS T1
INNER JOIN schools AS T2 ON T1.CDSCode =
T2.CDSCode WHERE T1.`Charter School (Y/N)` = 1
```

```
Schema Chunk 2 (3000 Tokens):
<TABLE>
schools (
 "CDSCode" TEXT,
 "ZIP" TEXT,
)
</TABLE>
```

```
Schema Chunk 1 (4000 Tokens):
<TABLE>
frpm (
 "CDSCode" TEXT,
 "Charter School (Y/N)" INTEGER,
...
 FOREIGN KEY("CDSCode")
REFERENCES schools ("CDSCode")
)
</TABLE>
...
```

```
Schema Link Chunk 2:
Tables: [schools]
Columns: [schools.Zip]
Foreign Keys: [ ]
```

```
Schema Link Chunk 1:
Tables: [ frpm ]
Columns: [frpm.`Charter School (Y/N)`]
Foreign Keys: [schools.CDSCode = frpm.CDSCode]
```

```
Schema Link Final:
Tables: [ schools, frpm]
Columns: [schools.Zip, frpm.`Charter School (Y/N)`]
Foreign Keys: [schools.CDSCode = frpm.CDSCode]
```

Fig. 5. Schema chunking process

model's task was to predict only the schema links present in the prompt and predict None if none of them were present.

3.5.2 Not trusting hypothesis. The training process begins by first training the schema linking model. After fine-tuning this model, we generated schema links for the entire data set. Subsequently, the model that generates the SQL is later trained using the schema link deduced by the first model and a resume version of the original schema. This approach offers the advantage of distributing the risk of inferring the structure of the database wrong to answer the question between the two models.

One notable limitation of this approach is that the schema link model generates better results for the training dataset, as the model has previously "seen" these examples, potentially biasing the SQL model to be more trustful of the schema link recommendations. However, due to the limited data, we decided to make this trade-off rather than further splitting the training dataset.

3.5.3 Trusting hypothesis. Both the Schema-Link model and the SQL model are trained using the schema links constructed using reverse engineering from the BIRD dataset. This creates a trusting scenario where the SQL model trusts the output given by the schema link model, focusing solely on creating a syntactically and logically correct SQL query.

Table 1. Results Vanilla models

Model	Simple	Moderate	Challenging	**Total**
$ND_p + V$	20.76%	5.81%	1.39%	**14.41%**
$ND_p + V_c$	23.24%	7.96%	4.17%	**16.82%**

Table 2. Results direct fine-tune

Model	Simple	Moderate	Challenging	**Total**
$ND_p + V$	20.76%	5.81%	1.39%	**14.41%**
$ND_p + V_c$	23.24%	7.96%	4.17%	**16.82%**
$ND_p + \text{SQL}_{ft}$	35.79%	15.05%	5.56%	**26.73%**

4 RESULTS

We tested various models and combinations, the following section describes each one of them and the results obtained. We used the Dev partition of the BIRD-dataset and used the official Bird code in order to evaluate our models.

4.1 Vanilla

First, we tested how the vanilla version of code-llama 13B would perform in the Bird-SQL dataset. In order to fit the schema and descriptions into the prompt we used a non-descriptive approach (ND_p), that would leave out the column descriptions if they didn't fit in the prompt. We used 5000 tokens as the threshold.

In the prompt was specified that we only wanted the SQL query with no explanation but sometimes the model would also include an explanation of the query against what was prompted.

This led us to divide the vanilla model into two results "Vanilla" (V), which evaluates results exactly as the model outputs them, and "vanilla clean" (V_c) where we employed a post-processing step to extract only the SQL query from the generated results.

Table 1 describes the execution accuracy results.

We conducted this test in order to understand the current capabilities of the models and set a floor value in order to improve upon it.

4.2 Zero shot fine-tune

Later we fine-tuned a single code-llama 13B model in order to predict the SQL answer in a zero-shot environment (SQL_{ft}). In this model, we again used a non-descriptive approach (ND_p) for the prompt as the only task required was to generate the final SQL query and this generation could not be divided into multiple prompts. The results are in the table 2. These results give us a comparison point between a one-step process and our approach of sub-dividing the problem into multiple steps.

Table 3. Results Non-Trusting

Model	Simple	Moderate	Challenging	**Total**
$ND_p + V$	20.76%	5.81%	1.39%	**14.41%**
$ND_p + V_c$	23.24%	7.96%	4.17%	**16.82%**
$ND_p + \text{SQL}_{ft}$	35.79%	15.05%	5.56%	**26.73%**
$ND_p + SL + NT$	42.16%	25.59%	19.44%	**35.01%**
$CH_p + SL + NT$	52.97%	39.35%	29.86%	**46.68%**

Table 4. Results Trusting

Model	Simple	Moderate	Challenging	**Total**
$ND_p + V$	20.76%	5.81%	1.39%	**14.41%**
$ND_p + V_c$	23.24%	7.96%	4.17%	**16.82%**
$ND_p + \text{SQL}_{ft}$	35.79%	15.05%	5.56%	**26.73%**
$ND_p + SL + NT$	42.16%	25.59%	19.44%	**35.01%**
$CH_p + SL + NT$	52.97%	39.35%	29.86%	**46.68%**
$CH_p + SL + T$	32.22%	27.20%	9.03%	**25.49%**

4.3 Non-Trusting

For our model combination, we first evaluated our non-trusting approach (NT). In this approach, the SQL model was trained with the schema links predicted by the Schema-Link model which were not always correct thus creating a non-trusting environment. This model was evaluated using a non-descriptive (ND_p) and a chunk approach (CH_p) for the schema link model.

In the chunk approach (CH_p), we divide the schema with the column descriptions in N prompts, each prompt having a maximum threshold length. We generate predictions for each prompt and later join them to create the final schema link prediction.

Initially, we generated schema link predictions for both the non-descriptive and chunked schema link models. Subsequently, we provided these predictions to the SQL model, along with the schema and column descriptions incorporating the latter when they could be accommodated within the context without exceeding it.

4.4 Trusting

In the "trusting" version of the model, we generated the schema link using the chunk approach immediately ($CH_p + SL + T$). Subsequently, only the question and this prediction were prompted to the SQL model.

4.5 Upper Bound

Finally, we tested our model's upper bound, specifically the SQL model's upper bound, we did this in order to further understand our model's limitations. To test this we prompted the SQL model with the perfect schema links SL_p extracted as detailed previously. This way the model would have the perfect recommendation of columns, tables and foreign keys to use.

We also tested how the non-trusting model would perform if only the schema link and external knowledge were prompted in order to not confuse the LLM with extra content, the extra information was not necessary given that the schema links are perfect (NT_{sl}), Table 5 details our results.

Table 5. Results Upper Bound

Model	Simple	Moderate	Challenging	**Total**
$SL_p + NT$	55.78 %	41.72%	29.86%	**49.09%**
$SL_p + T$	53.73%	29.03%	24.31%	**43.48%**
$SL_p + NT_{sl}$	35.46%	14.84%	11.81%	**26.99%**

4.6 Analysis

In this section, we are going to contrast the results of the different models to see if our hypothesis were correct or not.

4.6.1 Vanilla v/s Zero shot fine-tune. The best vanilla results were 16.82% ($ND_p + V_c$) and the worst of the fined-tuned models got 26.73%. This shows an improvement of ~10% proving the hypothesis that trained models such as Llama get much better results when fine-tuned and are a powerful tool in resolving text-to-SQL problems.

4.6.2 Trusting v/s Zero shot fine-tune. The Trusting version did not perform well, as we can see in Table 4, getting only 25.49% losing against the $ND_p + \text{SQL}_{ft}$ version.

This is due to the amount of responsibility that the schema linking step took. The input of the SQL model was solely the schema link, limiting the amount of context that the used model to create the SQL prediction. What happened in the majority of cases was that the SL failed to capture a column needed or captured the name wrong, giving zero chance to the next step to get the answer well. From this, we learned two things:

1. Getting the exact amount of information from the data schema that is needed to answer the question is not effective. It's better for the model to include extra columns than to risk having too few to make the right SQL prediction.
2. Prompting the schema to the SQL model gives the chance to fix the SL output, spreading the risk of not understanding the database architecture and enhancing the performance of the model greatly.

4.6.3 Non-Trusting v/s Zero shot fine-tune. Comparing the results between our non-trusting approach and a zero-shot fine tune we can see a big spike in performance, providing evidence that dividing the tasks between models is beneficial when creating SQL queries. The improvement was ~10% for a non-descriptive setup (similar comparisons) and a ~20% improvement for a chunked setup, providing further evidence that chunking the schema can lead to better results.

4.6.4 Trusting v/s Non-Trusting Upper Bound. To our surprise, the Non-trusting model performed better than the trusting one, this is a surprise because the trusting model was trained to create queries based solely on the schema links prompted to it, leaving it only the responsibility of creating SQL queries. A reason this could be the case is that after two epochs the trusting SQL model continued to learn i.e. the loss function on the validation set continually decreased and was stopped at the end of the second epoch.

To our surprise the model with just the schema linking (NT_{sl}) performed worse, this could be due to the fact that it was fine-tuned with a different prompt layout confusing the model more than helping it. One key takeaway is that fine-tuned models are very sensitive to prompt changes, further studies are required in order to explore this hypothesis.

5 CONCLUSION

Task division can be incredibly beneficial for LLM models, many agents and prompt chaining approaches have risen lately. In this study we proved how task division can be beneficial for NL2SQL tasks, increasing performance by more than 10 basis points compared to a direct approach. We also explored the idea and proposed a new framework to divide schema linking prompts in order to capture as much information as possible for the database schema and descriptions, this helped us improve by a further 10% when compared to the non-descriptive approach. Finally, we tried combining two different models in order to further enhance the performance. This study proves that smaller fine-tuned and task-specific approaches can deliver competitive results compared to GPT-4, it also presents a cheap way of fine-tuning models using easy access Google-colab notebooks.

6 LIMITATIONS

There are a few significant limitations of this study, first of all, we couldn't test the impact of having 2 different models each specialized in one task, we encourage others to further validate the hypothesis that using LLama 2 for the schema link and Code-Llama for the SQL generation is actually beneficial.

The second limitation we encountered was coming up with a chunking method that would prevent information loss. Sometimes during our chunking two tables joined by a Foreign key would get split up, although the table information containing the foreign key reference remained, the fact that the model didn't have access to the other table and its column could play an important part in picking the correct columns.

Finally, due to time limitations, we couldn't study what the new error composition looked like, this could lead to an interesting study of further modularizing the steps needed in order to generate a correct SQL.

7 ACKNOWLEDGMENTS

Special thanks to Hans Löbel for his constant help and guidance during this process.

REFERENCES

Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *arXiv preprint arXiv:2308.15363* (2023).

Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2022. Decomposed prompting: A modular approach for solving complex tasks. *arXiv preprint arXiv:2210.02406* (2022).

Jinyang Li, Binyuan Hui, Ge Qu, Binhua Li, Jiaxi Yang, Bowen Li, Bailin Wang, Bowen Qin, Rongyu Cao, Ruiying Geng, et al. 2023. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *arXiv preprint arXiv:2305.03111* (2023).

Mohammadreza Pourreza and Davood Rafiei. 2023. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *arXiv preprint arXiv:2304.11015* (2023).

Baptiste Rozière, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code llama: Open foundation models for code. *arXiv preprint arXiv:2308.12950* (2023).

Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288* (2023).

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-sql task. *arXiv preprint arXiv:1809.08887* (2018).

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *arXiv preprint arXiv:1709.00103* (2017).

Received 16 October 2023