The Death of Schema Linking? Text-to-SQL in the Age of Well-Reasoned Language Models

Karime Maamari[1] **Fadhil Abubaker**[1] **Daniel Jaroslawicz**[1] **Amine Mhedhbi**[2]
[1]Distyl AI [2]Polytechnique Montréal
{karime,fadhil,daniel}@distyl.ai
amine.mhedhbi@polymtl.ca

Abstract

In Text-to-SQL pipelines, schema linking is used to retrieve tables and columns that are relevant to the user's natural language query. However, inaccuracies in schema linking can lead to the exclusion of crucial information, which in turn adversely affects SQL generation. In this work, we revisit the need for schema linking when using the latest generation of large language models (LLMs). We find that newer models can accurately identify relevant schema during SQL generation, even in the presence of substantial irrelevant data. Consequently, our Text-to-SQL pipeline forgoes schema linking when the entire database schema fits within the model's context window. This approach eliminates errors due to faulty schema linking by ensuring that no schema information is omitted. Furthermore, we introduce techniques such as augmentation, selection, and correction, which improve Text-to-SQL accuracy without the risk of filtering out essential schema information. Our approach ranks first on the BIRD benchmark, achieving an accuracy of 71.83%.

1 Introduction

We address the task of Text-to-SQL: generating a database-executable SQL query given a natural language inquiry [2, 23]. Text-to-SQL is crucial in democratizing data access as it allows querying databases using natural language. The advent of large language models (LLMs) has significantly advanced Text-to-SQL by simplifying the translation of natural language into SQL.

LLM-based Text-to-SQL approaches typically follow a multi-stage generation pipeline [10, 14, 18, 31], as shown in Fig. 1. The pipeline begins with a retrieval stage to collect contextual knowledge such as the definition of terms and database schema elements. This is followed by a generation stage, where an LLM produces a candidate SQL query. Finally, the correction stage regenerates the SQL as needed based on encountered errors.

Within the retrieval stage, *schema linking* refers to the step of collecting relevant elements of the database schema such as tables and columns. It provides the necessary context for the LLM to produce correct SQL in the downstream generation stage and it is seen as the 'crux of the task' [13]. Effective schema linking implies retrieving *all* the relevant database components associated with the natural language query, as missing even a single required column produces incorrect SQL. However, while it is vital to ensure that all essential columns are retrieved, this does not mean that schema linking should be overly inclusive. Research has shown that the number of false positives, *i.e.*, irrelevant columns passed to the LLM, can degrade Text-to-SQL accuracy. For example, It has been shown that even when the entire database schema fits into the context of an LLM, it is still advantageous to perform schema linking [6]. Nevertheless, attempting to prune irrelevant columns through schema linking may also inadvertently eliminate essential ones. Thus, schema linking contains an inherent trade-off between minimizing false positives while also preserving relevant context.

Table Representation Learning Workshop at NeurIPS 2024.

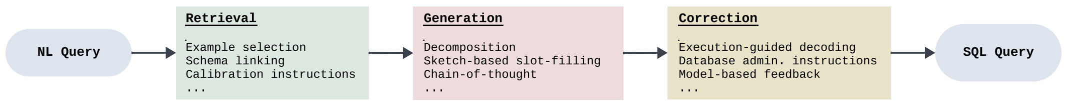


Figure 1: A typical Text-to-SQL pipeline comparised of retrieval, generation and correction stages.

As LLM reasoning capabilities improve, we challenge the conventional wisdom that schema linking is necessary for accurate Text-to-SQL when the schema fits within the model's context window. We find empirically that as model reasoning improves, the benefits of reducing false positives diminish; *i.e.*, newer models are more capable of sifting through context to identify relevant columns compared to older models [11, 16]. This effect is similar to how the latest LLMs when probed can reliably recall 'a needle in multiple millions of tokens of haystack' [1, 24]. For these models, schema linking is unnecessary and can even be detrimental, as it may filter out essential columns. Instead, we present alternatives to schema linking that improve accuracy without schema information loss. Our approach based on these insights currently ranks first in accuracy at 71.83% on the BIRD benchmark [15].

2 Related Work

2.1 Text-to-SQL Pipeline Overview

In current state-of-the-art approaches, Text-to-SQL uses multi-stage pipelines comprised of retrieval, generation, and correction stages.

The retrieval stage gathers relevant contextual information, including schema elements, domain knowledge, and example queries [5, 7, 12, 21, 27].

The generation stage often involves more than just producing a candidate SQL query associated given an input context. Rather, approaches frequently augment the generation process through techniques like decomposed generation [4, 20, 21, 27] and chain-of-thought prompting [30]. In addition, most approaches employ methods like self-consistency and multi-choice selection to produce multiple results, selecting the best outcome [5, 7, 12].

The retrieval and generation stages often contain various techniques chained together. These techniques can be broadly categorized as filtering or augmenting, *i.e.*, techniques can either strip away unnecessary contextual information or attempt to provide additional useful context.

Finally, the correction stage will often employ some combination of execution-based feedback [28, 17, 9, 19] or model-based feedback [26, 3, 28] to correct the generated query.

2.2 Schema Linking

Within the retrieval stage, schema linking leverages sophisticated prompting techniques to produce variable-length representations, hierarchically retrieve components, and iteratively process the schema [26, 5, 21, 12, 27, 7].

These techniques can vary in i) how they represent the schema and ii) how they perform linking on that representation. For instance, some may represent the schema in natural language, while others utilize code-like structures [7]. The approach to linking can also differ across techniques, with some directly filtering the schema [5, 12, 26], and others using intermediate representations to identify relevant tables and columns [22]. The choice of schema representation and linking strategy can have a significant influence on accuracy [7]. This variability underscores the importance of selecting an appropriate method tailored to the specific requirements of the task, as the degree of filtering can directly impact the loss of schema information incurred during the schema linking process.

Most prior investigations of schema linking regardless of approach have arrived at the same conclusion – schema linking yields meaningful gains in accuracy [8, 14, 26]. However, these explorations used LLMs that are more sensitive to the presence of irrelevant columns (false positives) as contextual information, where reducing the false positives yielded meaningful gains in performance [6].

3 Experimental Setup

3.1 Overview

Our experimental analysis guides the design and implementation of our proposed Text-to-SQL pipeline (§4.4). Our experiments aim to answer empirically three research questions (RQs):

RQ1. How does the inclusion of irrelevant schema elements impact SQL generation?
RQ2. How can the trade-off between precision and recall in schema linking techniques be characterized, and what is its downstream impact on SQL generation?
RQ3. How do other techniques and stages within Text-to-SQL pipelines, aside from schema linking, comparatively impact SQL generation?

3.2 Datasets

We conducted our experiments using the BIRD dataset [15], which is widely considered to be the most challenging Text-to-SQL benchmark. BIRD contains queries from 95 databases spanning a wide breadth of domains, such as education and hockey, and is designed to mimic the complexity of real-world databases. This complexity arises from its *"dirty"* format, where data, queries, and external knowledge may contain flaws — queries can be incorrect, database columns might be improperly described, and databases can contain null values and unexpected encodings. Our evaluation set consisted of 10% of the entries from each database in the dev set, as done in evaluations in prior work [26]. Our training set consisted of 500 of the $9,428$ available samples in the BIRD training dataset.

3.3 Models

We used the following language models with context windows sufficiently large to accommodate the entire schema for each query in the BIRD evaluation set:

`ft:GPT-4o (fine-tuned)`	`Llama 3.1-405b`	`Claude 3.5 Sonnet`
`GPT-4o`	`Llama 3.1-70b`	`Claude 3 Opus`
`GPT-4o-Mini`	`Llama 3.1-8b`	`Mixtral-8x22B`
`GPT-4-Turbo`	`Deepseek Coder-V2`	`Gemini 1.5 Pro`

We overview the fine-tuning approach of `GPT-4o` momentarily under methodology (§3.4).

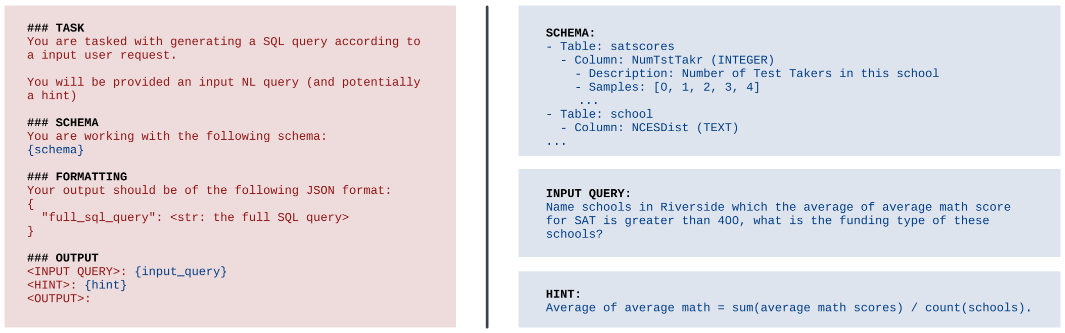
3.4 Methodology

We design an empirical experiment for each research question:

Exp 1. We use a simplified Text-to-SQL pipeline consisting of only schema linking within the retrieval stage followed by a single attempt at generation. For each query, we provide all the required columns and vary the amount of irrelevant columns to look at the impact of irrelevant columns retrieved on the generation.
Exp 2. Using the same simplified pipeline, we introduce different implementations of schema linking that vary in precision and recall and analyze their impact on generation.
Exp 3. We introduce augmentation, selection, and correction techniques on top of the simplified pipeline without and with schema linking. We run an ablation study to understand the relative impact of each technique on end-to-end accuracy.

Runs and input/output structure. In all runs, the temperature was set to zero and structured output was used whenever possible. Given that not all models are capable of reliable structured output generation, the generated SQL query was fed through an identity call by `GPT-4o-Mini` in `JSON` mode to handle any potential issues with output formatting. The relative position of any schema element in an input prompt follows the same ordering as that provided by the schema definition of the benchmark.

Fine-tuning `GPT-4o`. Fine-tuning is done iteratively. At each iteration, we first fine-tune on a sample of N triples: natural language query, SQL query, and schema elements. For each query, the schema includes all required columns and a random number of irrelevant columns picked uniformly at



```
### TASK
You are tasked with generating a SQL query according to
a input user request.

You will be provided an input NL query (and potentially
a hint)

### SCHEMA
You are working with the following schema:
{schema}

### FORMATTING
Your output should be of the following JSON format:
{
  "full_sql_query": <str: the full SQL query>
}

### OUTPUT
<INPUT QUERY>: {input_query}
<HINT>: {hint}
<OUTPUT>:
```

```
SCHEMA:
- Table: satscores
  - Column: NumTstTakr (INTEGER)
    - Description: Number of Test Takers in this school
    - Samples: [0, 1, 2, 3, 4]
    ...
- Table: school
  - Column: NCESDist (TEXT)
...
```

```
INPUT QUERY:
Name schools in Riverside which the average of average math score
for SAT is greater than 400, what is the funding type of these
schools?
```

```
HINT:
Average of average math = sum(average math scores) / count(schools).
```

Figure 2: **(Red)** Structure of SQL Generation prompt given input query, hint, and schema; **(Blue)** Examples of a schema, input query, and query hint which act as contextual inputs to a prompt.

random. We then evaluate on BIRD's dev set. For each failed query, we prompt the model to reason about the failure, aggregate the reasoning across queries, and use it to pick the sample for the next iteration. Finally, based on the reasoning, we select a new sample of size N. The iterations stop once a pre-determined accuracy score is reached.

Generation prompts. Fig. 2 shows the structure of the prompt used for SQL generation as well as an example schema, input query, and query hint.

3.5 Metrics

We rely on three different metrics in our experimental analysis:

- *Execution Accuracy* (EX): The metric used by the BIRD benchmark to evaluate end-to-end Text-to-SQL pipelines. It is the proportion of queries for which the output of the predicted SQL query is identical to that of the ground truth SQL query. We report EX as a percentage over queries in the evaluation set.
- *False Positive Rate* (FPR): For a given query, the proportion of irrelevant schema columns retrieved over the total number of retrieved columns. We report its average over queries in the evaluation set.
- *Schema Linking Recall* (SLR): The proportion of queries for which all required columns are retrieved over the total number of queries. We use SLR as the downstream generation requires all required columns to be retrieved to be correct.

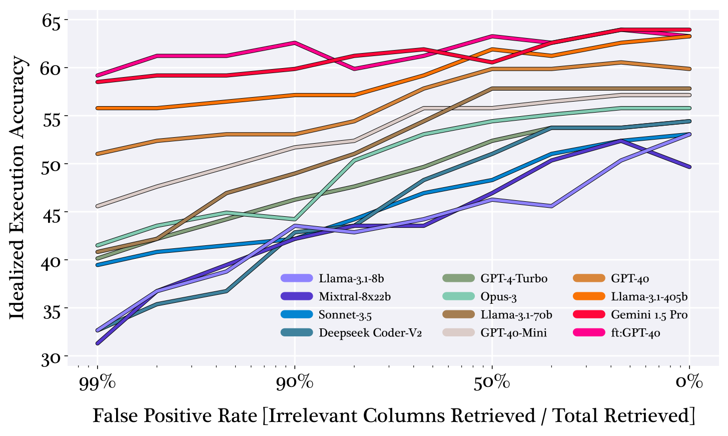
All queries in evaluation sets are across multiple databases. Note that BIRD also has a second metric: valid efficiency score. It assesses the efficiency of correctly predicted queries by comparing their execution speed to those of the corresponding ground truth queries. In our experiments, we focus on EX, as our research questions are primarily concerned with SQL generation accuracy.

4 Results

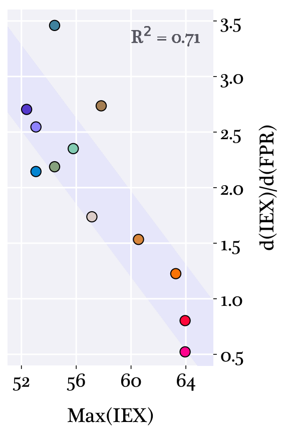
4.1 Experiment 1: Impact of False Positives on Accuracy Given Perfect SLR

In this first experiment, we assess the impact of retrieving irrelevant columns on SQL generation accuracy. We create a scenario with perfect schema linking recall such that SQL generation issues are not due to missing required columns.

To implement the experiment, we mock the schema linker as follows. We build an oracle that identifies all columns necessary for a given query using `SQLGLOT` (SQL Parser and Transpiler). For an input query, the mocked schema linker uses the oracle to retrieve all required columns and injects a pre-defined rate of false positives (irrelevant columns retrieved / total columns retrieved). To inject it, the mocked schema linker samples the irrelevant columns uniformly at random from the target database. If there is not a sufficient number of columns in the target database, *e.g.*, there are 10 required columns and 900 irrelevant columns are needed to produce a 99% false positive rate, yet only 50 columns exist in the target database, it supplements from other databases with columns not



(a) The idealized execution accuracy (IEX) as the false positive rate (FPR) varies from 99% to 0%.



(b) Capability and sensitivity relationship.

Figure 3: Idealized execution accuracy (IEX), *i.e.*, with perfect schema linking recall (SLR), as the false positive rate (FPR) varies for different LMs and the relationship between their capability (maximum IEX) and sensitivity to false positives.

conflicting in name. We use a simple pipeline: a retrieval stage containing only the mocked schema linker followed by a zero-shot generation stage.

We run the pipeline using the 12 selected models (§3.3) for generation while applying an equal false positive rate to each query in the evaluation set. We run for 10 different rates that are equally log-spaced from 99% and 0%. We refer to the execution accuracy under perfect schema linking recall as the idealized execution accuracy (IEX). We track the change in IEX as the false positive rate changes. The results in Fig. 3a show the broad trend that IEX improves as the rate of false positives decreases; specifically, the stage of SQL generation improves as less irrelevant columns are included as contextual information. At one extreme with a maximum rate of false positives (99%), there is a $\sim$ 28% IEX difference between the worst and best performing models. In the other extreme with no false positives (0%), the IEX difference between the worst and best performing models gets reduced to $\sim$ 14%.

We use a model's maximum IEX in this experiment as a proxy for its SQL generation capability. Models with higher generation capability, such as `Gemini 1.5 Pro`, demonstrate greater resilience to false positives compared to lower-performing ones like `Llama 3.1-8b`. We characterize a model's resilience to false positives by the relative change in its SQL generation capability as the false positive rate changes. We define a metric for sensitivity to false positives as the proportion of change in IEX over the proportion of change in the false positive rate (FPR). As such, sensitivity to false positives is the slope derived from the model's (IEX, FPR) data points. Fig. 3b depicts a strong negative correlation between a model's SQL generation capability (max IEX) and its sensitivity to the FPR.

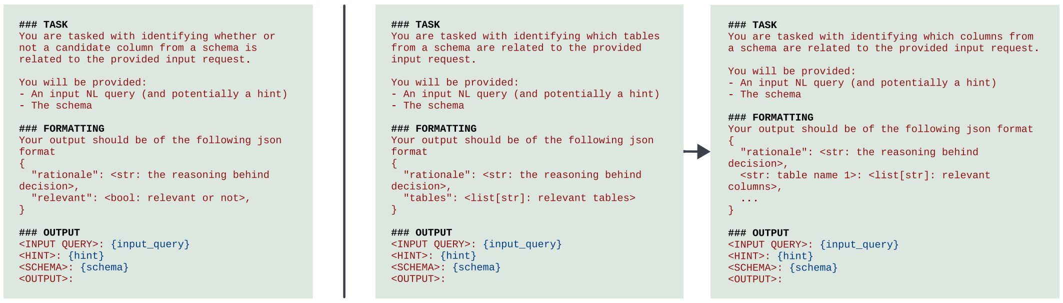
Empirical Observation: As the model's SQL generation capability improves, its sensitivity to the presence of irrelevant columns as contextual information for generation decreases. Perhaps surprisingly, both of these capabilities go hand-in-hand where models that are generally better at SQL generation are also more resilient to large amounts of irrelevant context.

4.2 Experiment 2: Impact of False Positives on Accuracy Given Actual SLR

In Experiment 1, all necessary columns for generation are provided, regardless of the false positive rate. However, in practice, decreasing the amount of false positives requires pruning, which carries the risk of excluding required columns. Here we assess the extent to which schema linking affects recall of required columns and the downstream impact of imperfect recall on generation.

We explore four different schema linking approaches with a broad spectrum of ability to reduce the false positive rate:

- *Single-Column Schema Linking* (SCSL): Model-determined column-wise relevance. The relevance of each column is assessed without context about other columns and tables. The output is



```
### TASK
You are tasked with identifying whether or
not a candidate column from a schema is
related to the provided input request.

You will be provided:
- An input NL query (and potentially a hint)
- The schema

### FORMATTING
Your output should be of the following json
format
{
  "rationale": <str: the reasoning behind
decision>,
  "relevant": <bool: relevant or not>,
}

### OUTPUT
<INPUT QUERY>: {input_query}
<HINT>: {hint}
<SCHEMA>: {schema}
<OUTPUT>:
```

```
### TASK
You are tasked with identifying which tables
from a schema are related to the provided
input request.

You will be provided:
- An input NL query (and potentially a hint)
- The schema

### FORMATTING
Your output should be of the following json
format
{
  "rationale": <str: the reasoning behind
decision>,
  "tables": <list[str]: relevant tables>
}

### OUTPUT
<INPUT QUERY>: {input_query}
<HINT>: {hint}
<SCHEMA>: {schema}
<OUTPUT>:
```

```
### TASK
You are tasked with identifying which columns from
a schema are related to the provided input request.

You will be provided:
- An input NL query (and potentially a hint)
- The schema

### FORMATTING
Your output should be of the following json format
{
  "rationale": <str: the reasoning behind
decision>,
  <str: table name 1>: <list[str]: relevant
columns>,
  ...
}

### OUTPUT
<INPUT QUERY>: {input_query}
<HINT>: {hint}
<SCHEMA>: {schema}
<OUTPUT>:
```

Figure 4: Prompts used for schema linking. **(Left)** Single-Column Schema Linking (SCSL): identifying relevance of a particular column independent of the rest of the schema; **(Middle + Right)** Table-to-Column Schema Linking (TCSL): first identifying relevant tables then relevant columns.

a boolean flag per column indicating the relevance of each column. This is considered a more cautious approach less likely to filter out relevant columns.

- *Hybrid SCSL* (HySCSL): SCSL with added keyword matching aiming for higher SLR.
- *Table-then-Column Schema Linking* (TCSL): Model-determined table-then-column filtering approach, as proposed in [26] and [21]. The model first filters the schema to the relevant tables, then filters the columns within those tables. The output is a set of relevant columns and tables. This is a more aggressive filtering approach.
- *Hybrid TCSL* (HyTCSL): TCSL with added keyword matching aiming for higher SLR.

In our implementation, SCSL and HySCSL use `GPT-4o-Mini` and TCSL and HyTCSL use `GPT-4o`. We use `GPT-4o-Mini` with SCSL and HySCSL as `GPT-4o` shows a negligible gain in our experiments but is much cheaper. The prompts used in our implementation are shown in Fig. 4.

Table 1 reports the mean (± stddev) of the false positive rate (FPR) and the schema linking recall (SLR) across our four schema linking techniques and without schema filtering at all (full schema) from 12 different runs. Table 1 shows that these approaches are robust across runs and that as FPR decreases, *i.e.*, more irrelevant columns are filtered, more required columns can be filtered and SLR decreases. SLR as obtained from schema linking represents an upper bound on possible EX, where $100\% - SLR$ represents the ratio of queries that will fail in the generation stage due to missing required columns.

We run the same simplified pipeline as our first experiment: a retrieval stage containing one of the five schema linking approaches in Table 1 followed by a zero-shot generation stage. We do so across the 12 selected models (§3.3) and track the associated EX. Fig. 5 shows five EX data points given the FPR of the five different schema linking approaches. We interpolate between two adjacent (FPR, EX) data points and show EX as a solid line. We also add the idealized EX from Experiment 1 as a dashed line. The difference between the two lines shows the EX difference due to changes in SLR.

We observe three different classes of models in our results: models where some schema linking method improves performance (*e.g.*, `Llama 3.1-8b`), models where all schema linking methods degrade performance (*e.g.*, `Gemini 1.5 Pro`), and models where schema linking has only negligible impact on performance (*e.g.*, `GPT-4o-Mini`). A model falls into one of these three buckets based on its SQL generation capability. More capable models – e.g. `Gemini 1.5 Pro`, `ft:GPT-4o`, `Llama-3.1-405b` – end up with a reduction in EX. Less capable models – e.g. `Llama 3.1-8b`, `Mixtral-8x22b`, `Deepseek Coder-V2` – end up with a gain in EX.

Empirical Observation: As the model's SQL generation capability improves, the benefit of schema linking diminishes. In some cases, this can result in a net reduction in accuracy due to missing required columns for generation.

4.3 Experiment 3: Impact of Non-Filtering Stages and Techniques

Instead of filtering contextual information through schema linking, we focus on techniques that preserve information. We assess the gains of using *augmentation* and *selection* techniques as well as adding a correction stage, which are detailed as follows:

Approach	FPR	SLR
Without schema filtering (Full Schema)	94.62	100.00
Hybrid Single-Column Schema Linking (HySCSL)	82.08 ± 0.44	90.36 ± 0.78
Single-Column Schema Linking (SCSL)	67.23 ± 0.92	88.77 ± 0.75
Hybrid Table-to-Column Schema Linking (HyTCSL)	19.85 ± 0.99	83.00 ± 0.92
Table-to-Column Schema Linking (TCSL)	9.79 ± 0.73	77.44 ± 1.34

Table 1: The mean (± stddev) false positive rate (FPR) and schema linking recall (SLR), across 12 runs, associated with five schema linking approaches. The approaches are sorted in descending order of their ability to reduce false positives. We report the mean values (± stddev) from 12 runs.

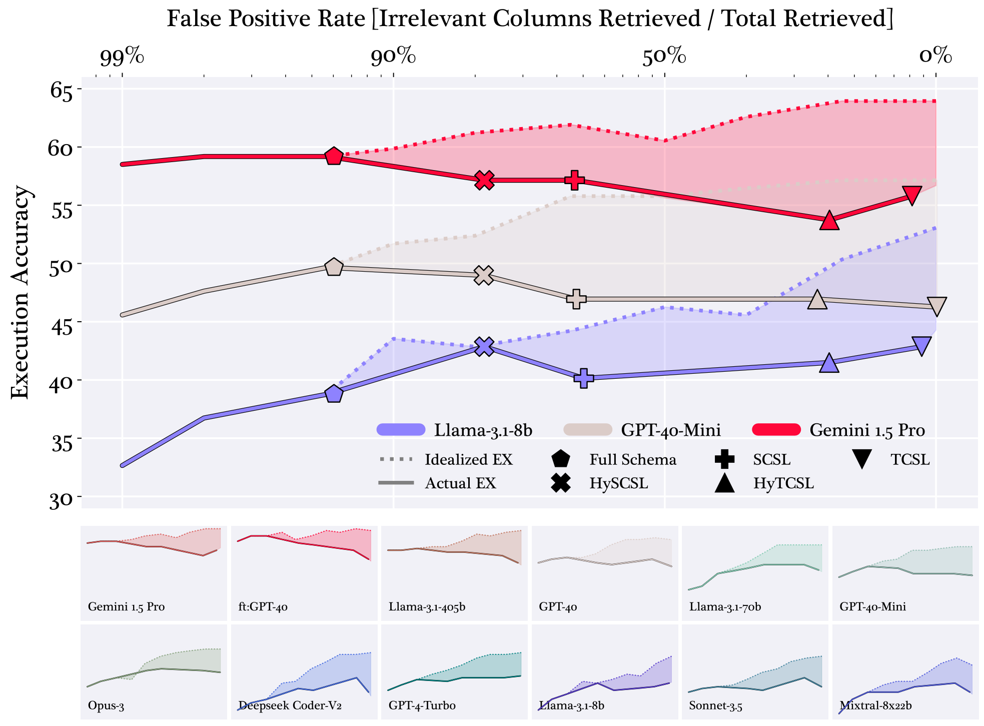


Figure 5: The execution accuracy (EX) given different schema linking and idealized EX, assuming perfect Schema Linking Recall (SLR) as the false positive rate (FPR) varies. EX as a solid line and idealized EX as a dashed line.

- *Augmentation:* We add contextual information by: (i) expanding column descriptions and add query hints and (ii) adding structural expectations of the output, *e.g.*, expected orderings and aggregations, using CoT planning.
- *Correction:* After generating a candidate SQL query, we iteratively apply corrections through re-generation based on database execution errors [28], revision through database administrator instructions [26], and model-based feedback similar to Reflexion [25]. We use these corrections to generate instructions to also augment contextual information.
- *Selection:* We use self-consistency [29] to generate multiple responses and select the *most consistent* result. We use selection across the whole pipeline for augmentation, SQL generation, and SQL correction.

We implemented a full pipeline using zero-shot generation. The pipeline contained augmentation, correction and selection as described above and no schema linking, *i.e.*, always providing the full schema. We ran an ablation to understand the relative impact of each technique or stage using the three top performing models from Experiment 1: `Llama 3.1-405b`, `Gemini 1.5 Pro`, and `ft:GPT-4o`. Table 2 shows the execution accuracy (EX) of the full pipeline and 6 other variations: without augmentation, correction, or selection, with schema linking (TCSL and SCSL from Experiment 2),

Method	**Execution Accuracy (EX)**		
	ft:GPT-4o	**Gemini 1.5 Pro**	**Llama 3.1-405b**
Full pipeline	67.35	60.54	59.18
w/o Augmentation	64.63 (↓ 2.72)	60.54	59.86 (↑ 0.68)
w/o Selection	65.31 (↓ 2.04)	57.82 (↓ 2.72)	58.50 (↓ 0.68)
w/o Correction	65.99 (↓ 1.36)	57.14 (↓ 3.40)	55.78 (↓ 3.40)
w/ TCSL	62.58 (↓ 4.77)	55.78 (↓ 4.76)	56.46 (↓ 2.72)
w/ SCSL	55.78 (↓ 11.57)	55.10 (↓ 5.44)	54.42 (↓ 4.76)
Base model	59.18 (↓ 8.17)	57.82 (↓ 2.72)	53.74 (↓ 5.44)

Table 2: Ablation of different methods reporting the execution accuracy (EX) for fine-tuned GPT-4o, Gemini 1.5 Pro, and Llama 3.1-405b. The table compares the full pipeline to variations without augmentation, selection, and correction techniques; with Table-to-Column Schema Linking (TCSL) and Single-Column Schema Linking (SCSL); and as base model performance. Reductions (↓) or increases (↑) in accuracy compared to the full pipeline are indicated.

and without any of these techniques *i.e.*, providing the full schema alone to a base model. We find that all techniques improve accuracy to varying degrees except schema linking with a noticeable difference when evaluating the full pipeline.

Empirical Observation: Each of augmentation, selection, and correction have a noticeable positive impact on generation accuracy. Note that even though base models such as `Gemini 1.5 Pro` may show comparable performance to a fine-tuned `GPT-4o` in SQL generation, they differ when evaluated within end-to-end Text-to-SQL pipelines. For instance, augmentation leads to major benefits with `GPT-4o` when compared with `Gemini 1.5 Pro`. An interesting research direction is understanding whether the relative benefits from augmentation between models hold across other tasks as well.

4.4 Discussion and Proposed Approach

Our empirical observations reveal several key insights that guide our proposed approach.

First, as a model's SQL generation capability improves, its ability to retrieve relevant schema elements from a full schema in its input context also improves. As such, schema linking for state-of-the-art LLMs is less important if the schema fully fits within the context window. However, it is still helpful for models with lower SQL generation accuracy as they struggle with false positives. In our approach, *we maximize the use of the LLM context window to minimize filtering required columns*.

Second, we find that combining augmentation, selection, and correction techniques heavily impacts the accuracy of a Text-to-SQL pipeline. However, the impact is not the same when evaluated in an end-to-end fashion even when comparing models with similar generation capability. In our approach, *we adopt augmentation, selection, and correction as detailed from the implementation in Experiment 3*. We further use `ft:GPT-4o` as the model of choice for generation since it provides the best end-to-end accuracy and makes the best use of augmentation.

These design choices yield an approach that ranks first on execution accuracy and second on valid efficiency score on the BIRD benchmark.

5 Conclusion

Is it the death of schema linking? For state-of-the-art models, if the schema fits within the context length – yes. However, for smaller or prior generation models, the accuracy gains from schema linking often justify the potential loss. Additionally, in real-world data-warehousing scenarios, the entire schema often exceeds the context window, requiring a multi-stage information retrieval pipeline. In such cases, we argue for maximally using the LLM context window in picking the top-K relevant columns to retain the necessary schema elements. We conclude that while the need for schema linking is highly use-case dependent, its importance is diminishing as costs decrease, context windows widen, and generation capabilities improve.

References

[1] 2023. `https://github.com/gkamradt/LLMTest_NeedleInAHaystack/blob/main/README.md`.

[2] I. Androutsopoulos, G. D. Ritchie, and P. Thanisch. Natural language interfaces to databases - an introduction. *CoRR*, abs/cmp-lg/9503016, 1995.

[3] A. Askari, C. Poelitz, and X. Tang. Magic: Generating self-correction guideline for in-context text-to-sql. *CoRR*, abs/2406.12692, 2024.

[4] N. Deng, Y. Chen, and Y. Zhang. Recent advances in text-to-sql: A survey of what we have and what we expect. *COLING*, 2022.

[5] X. Dong, C. Zhang, Y. Ge, Y. Mao, Y. Gao, lu Chen, J. Lin, and D. Lou. C3: Zero-shot text-to-sql with chatgpt. *CoRR*, abs/2307.07306, 2023.

[6] A. Floratou, F. Psallidas, F. Zhao, S. Deep, G. Hagleither, W. Tan, J. Cahoon, R. Alotaibi, J. Henkel, A. Singla, A. V. Grootel, B. Chow, K. Deng, K. Lin, M. Campos, K. V. Emani, V. Pandit, V. Shnayder, W. Wang, and C. Curino. Nl2sql is a solved problem... not! In *CIDR*, 2024.

[7] 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. *CoRR*, abs/2308.15363, 2023.

[8] 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. *CoRR*, abs/1905.08205, 2019.

[9] P. He, Y. Mao, K. Chakrabarti, and W. Chen. X-sql: reinforce schema representation with context. *CoRR*, abs/1908.08113, 2019.

[10] Z. Hong, Z. Yuan, Q. Zhang, H. Chen, J. Dong, F. Huang, and X. Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *CoRR*, abs/2406.08426, 2024.

[11] P. Laban, A. R. Fabbri, C. Xiong, and C.-S. Wu. Summary of a haystack: A challenge to long-context llms and rag systems. *CoRR*, abs/2407.01370, 2024.

[12] D. Lee, C. Park, J. Kim, and H. Park. Mcs-sql: Leveraging multiple prompts and multiple-choice selection for text-to-sql generation. *CoRR*, abs/2405.07467, 2024.

[13] W. Lei, W. Wang, Z. Ma, T. Gan, W. Lu, M. Kan, and T. Chua. Re-examining the role of schema linking in text-to-sql. 2020.

[14] B. Li, Y. Luo, C. Chai, G. Li, and N. Tang. The dawn of natural language to sql: Are we fully ready? *CoRR*, abs/2406.01265, 2024.

[15] J. Li, B. Hui, G. Qu, J. Yang, B. Li, B. Li, B. Wang, B. Qin, R. Cao, R. Geng, N. Huo, X. Zhou, C. Ma, G. Li, K. C. C. Chang, F. Huang, R. Cheng, and Y. Li. Can llm already serve as a database interface? a big bench for large-scale database grounded text-to-sqls. *CoRR*, abs/2305.03111, 2023.

[16] M. Li, S. Zhang, Y. Liu, and K. Chen. Needlebench: Can llms do retrieval and reasoning in 1 million context window? *CoRR*, abs/2407.11963, 2024.

[17] X. V. Lin, R. Socher, and C. Xiong. Bridging textual and tabular data for cross-domain text-to-sql semantic parsing. *CoRR*, abs/2012.12627, 2020.

[18] X. Liu, S. Shen, B. Li, P. Ma, R. Jiang, Y. Luo, Y. Zhang, J. Fan, G. Li, and N. Tang. A survey of nl2sql with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109, 2024.

[19] Q. Lyu, K. Chakrabarti, S. Hathi, S. Kundu, J. Zhang, and Z. Chen. Hybrid ranking network for text-to-sql. *CoRR*, abs/2008.04759, 2020.

[20] K. Maamari and A. Mhedhbi. End-to-end text-to-sql generation within an analytics insight engine. *CoRR*, abs/2406.12104, 2024.

[21] M. Pourreza and D. Rafiei. Din-sql: Decomposed in-context learning of text-to-sql with self-correction. *CoRR*, abs/2304.11015, 2023.

[22] 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. *CoRR*, abs/2405.15307, 2024.

[23] A. Quamar, V. Efthymiou, C. Lei, and F. Özcan. Natural language interfaces to data. *Found. Trends Databases*, 2022.

[24] M. Reid, N. Savinov, D. Teplyashin, D. Lepikhin, T. P. Lillicrap, J. Alayrac, R. Soricut, A. Lazaridou, O. Firat, J. Schrittwieser, I. Antonoglou, R. Anil, S. Borgeaud, A. M. Dai, K. Millican, E. Dyer, M. Glaese, T. Sottiaux, B. Lee, F. Viola, M. Reynolds, Y. Xu, J. Molloy, J. Chen, M. Isard, P. Barham, T. Hennigan, R. McIlroy, M. Johnson, J. Schalkwyk, E. Collins, E. Rutherford, E. Moreira, K. Ayoub, M. Goel, C. Meyer, G. Thornton, Z. Yang, H. Michalewski, Z. Abbas, N. Schucher, A. Anand, R. Ives, J. Keeling, K. Lenc, S. Haykal, S. Shakeri, P. Shyam, A. Chowdhery, R. Ring, S. Spencer, E. Sezener, and et al. Gemini 1.5: Unlocking multimodal understanding across millions of tokens of context. *CoRR*, abs/2403.05530, 2024.

[25] N. Shinn, F. Cassano, E. Berman, A. Gopinath, K. Narasimhan, and S. Yao. Reflexion: Language agents with verbal reinforcement learning. *CoRR*, abs/2303.11366, 2023.

[26] S. Talaei, M. Pourreza, Y.-C. Chang, A. Mirhoseini, and A. Saberi. Chess: Contextual harnessing for efficient sql synthesis. *CoRR*, abs/2405.16755, 2024.

[27] B. Wang, C. Ren, J. Yang, X. Liang, J. Bai, L. Chai, Z. Yan, Q.-W. Zhang, D. Yin, X. Sun, and Z. Li. Mac-sql: A multi-agent collaborative framework for text-to-sql. *CoRR*, abs/2312.11242, 2024.

[28] C. Wang, K. Tatwawadi, M. Brockschmidt, P.-S. Huang, Y. Mao, O. Polozov, and R. Singh. Robust text-to-sql generation with execution-guided decoding. *CoRR*, abs/1807.03100, 2018.

[29] X. Wang, J. Wei, D. Schuurmans, Q. Le, E. Chi, S. Narang, A. Chowdhery, and D. Zhou. Self-consistency improves chain of thought reasoning in language models. *CoRR*, abs/2203.11171, 2023.

[30] J. Wei, X. Wang, D. Schuurmans, M. Bosma, B. Ichter, F. Xia, E. Chi, Q. Le, and D. Zhou. Chain-of-thought prompting elicits reasoning in large language models. *CoRR*, abs/2201.11903, 2023.

[31] W. Zhang, Y. Wang, Y. Song, V. J. Wei, Y. Tian, Y. Qi, J. H. Chan, R. C.-W. Wong, and H. Yang. Natural language interfaces for tabular data querying and visualization: A survey. *CoRR*, abs/2310.17894, 2024.