In-depth Analysis of LLM-based Schema Linking

George Katsogiannis-Meimarakis
IBM Research, Switzerland
Univ. Grenoble Alpes, France
Athena Research Center, Greece
katso@athenarc.gr

Katsiaryna Mirylenka
IBM Research
Zurich, Switzerland
kmi@zurich.ibm.com

Paolo Scotton
IBM Research
Zurich, Switzerland
psc@zurich.ibm.com

Francesco Fusco
IBM Research
Zurich, Switzerland
ffu@zurich.ibm.com

Abdel Labbi
IBM Research
Zurich, Switzerland
abl@zurich.ibm.com

Abstract

This work examines the critical component of schema linking in the Text-to-SQL domain, investigating the ability of Large Language Models (LLMs) to accurately identify relevant database tables and columns for SQL query generation from natural language requests. We conduct an in-depth analysis of LLM-based schema linking approaches, exploring the best practices to obtain high-quality predictions. Additionally, we experiment with multiple techniques, such as question decomposition, that have been successfully applied in Text-to-SQL and test their benefits to schema linking. We also challenge the prevailing assumption that the oracle linked schema, comprising the minimum set of columns necessary for SQL generation, is always the optimal schema representation. Our experiments on the Spider and BIRD benchmarks demonstrate the ability of LLMs to perform high-quality schema linking, boosting the overall Text-to-SQL performance.

Keywords

Schema Linking, Text-to-SQL, NLIDBs, LLMs

1 Introduction

The demand for natural language interfaces to relational databases (Text-to-SQL) has grown significantly, particularly among non-technical users. This longstanding problem, which has been explored for decades [5, 10, 43], has gained renewed interest in recent years due to the rapid advancements in natural language processing (NLP) models. However, this task remains a significant challenge due to the complexity of both natural language understanding and relational database schema navigation [4, 21, 23, 37, 38]. Given the difficulty of the problem, researchers are also focusing on breaking it down and tackling its sub-problems [8, 15, 19, 30, 35, 41, 48]. One such sub-problem is schema linking, which is the focus of this work.

Schema Linking is a component of Text-to-SQL systems that, given a Natural Language Question (NLQ), aims at identifying the DB elements (tables and columns) that are necessary to transform the NL question into its corresponding SQL representation [25, 50]. Modern relational databases (RDBs) can encompass a multitude of tables and columns, making it difficult for Text-to-SQL systems to simultaneously identify the relevant elements and generate accurate SQL queries. Schema linking offers two significant advantages for Text-to-SQL models: (i) it relieves the model from the burden of identifying relevant database elements, enabling it to concentrate solely on generating the correct SQL structure, and (ii) it reduces the number of database tables and columns that need to be included in the prompt, cutting cost and processing time. The second benefit is crucial, as real-world databases often contain an overwhelming number of tables that exceed the prompt length of certain models.

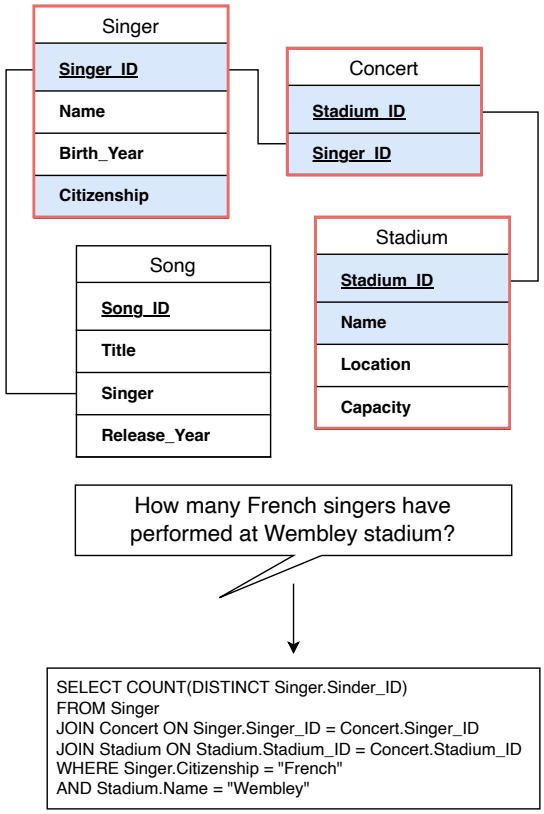


Figure 1: An example of schema linking. The highlighted blue columns and red tables are necessary for translating the question to SQL.

Figure 1 show an example of the schema linking task where given a DB schema describing Singers, Concerts, Songs and Stadiums, and a user question: 'How many French singers have performed at Wembley stadium? ' the task is to identify the relevant schema elements: 'Singer.Singer_ID', 'Singer.Citizenship', 'Concert.Singer_ID', 'Concert.Stadium_ID', 'Stadium.Stadium_ID' and 'Stadium.Name'.

EDBT '26, Tampere (Finland)

Earlier works that used schema linking [7, 46] mostly relied on n-gram comparison and string matching between the NLQ and the DB tables and columns. These approaches were relatively simple and could discover links when the user referred to the DB elements explicitly and with the same vocabulary as in the DB declarations. Recent works are using dedicated models for schema linking that are more robust to understanding synonyms and discovering implicit mentions in the NLQ. While some systems use ranking approaches [26], or generate a preliminary SQL query for schema linking, most works [12, 35, 49] take advantage of LLMs to generate the needed tables and columns, which are the main focus of this work.

Although multiple previous works have used LLMs as schema linking components, showcasing their importance, a systematic analysis of their performance, their sensitivity to hyperparameters such as the number of demonstration examples, and the best practices to use them still remain underexplored. In fact, most systems propose to prompt an LLM with the schema and the NLQ and instruct it to choose the right tables and columns for schema linking, without evaluating its performance on schema linking or investigating if it could be improved. Furthermore, most of these works rely on closed and proprietary LLMs. such as GPT-4 [32], making the reproducibility of their results unreliable due to undisclosed changes that are often performed by their creators.

In this work, we perform an experimental analysis of LLM-based schema linking, evaluating the performance and trade-offs of different models for the schema linking task and their effect on the text-to-SQL task. Given that LLM-based schema linking is usually not clearly defined or formalized in previous works, we created a unified schema linking pipeline that allows us to fairly compare different LLMs. Furthermore, we use techniques that have been proven to work well in LLM-based text-to-SQL and test them in the context of schema linking. Specifically, we experiment with in-context learning with demonstration retrieval, supervised fine-tuning, and question decomposition to uncover the best practices for LLM-based schema linking. Furthermore, we investigate whether common drawbacks of LLMs, such as hallucinations and lack of DB knowledge, can be easily remediated by a set of simple refinement techniques. Finally, we use a schema enrichment technique to test the common notion that perfect schema linking (i.e. only keeping the bare minimum necessary tables/columns) leads to the best performance, or if text-to-SQL models benefit from additional schema info that is not used in the final SQL.

Our experimental analysis provides multiple insights on the best practices for LLM-based schema linking and its effect on Text-to-SQL performance. The main contributions and the outline of our work are the following: Section 2 presents the related work to our study and an overview of previous studies and approaches to schema linking. Section 4 presents the methodology that was used to implement the pipeline for our experiments. Sections 5, 6, and 7 present our experimental results, offering valuable insights into the current schema linking capabilities of LLMs and revealing insights on how to best use them. Our code is made publicly available[1].

2 Related Work

2.1 Text-to-SQL and Previous Surveys

The field of Text-to-SQL has seen a large increase in research and industrial interest during the past years, with major driving forces being the introduction of large cross-domain benchmarks, such as Spider [51] and BIRD [28], and the advancement of NLP techniques such as LLMs based on the Transformer [44].

Due to the great interest shown for the Text-to-SQL problem and the numerous systems proposed in the literature, there have been multiple works that have provided literature reviews and surveys of existing solutions. Earlier works [2, 24] focused mostly on more traditional approaches from the DB community that did not rely on neural networks. Later surveys [11, 21] investigated the use of neural networks for the problem, which at the time focused on introducing novel architectures and representation techniques [6, 46, 53]. Several latest surveys [20, 55] provide an overview of Text-to-SQL approaches using LLMs, classifying them into categories based on their training and usage. The latest works [12, 35, 45, 49], that are based on LLMs, focus on breaking down the Text-to-SQL problem into smaller steps and solving them separately, and improving the quality of their input instructions (i.e.,prompts) and of the generated SQL queries. Our work differs from these previous surveys in the form that it focuses on experimental evaluation of schema linking, while they only briefly study it as a part of the larger Text-to-SQL pipeline, without any experiments.

Another set of recent works [9, 16, 27, 52] study how to create better prompts for the LLM that contain helpful information (e.g.,schema information, column types, more informative table and column names, etc.) as well as techniques for retrieving the most relevant few-shot demonstrations [16, 27] for In-Context Learning (ICL). Our work takes advantage of these studies by using prompting and retrieval techniques that have already been proven to work well, letting us focus on the schema linking problem.

2.2 Schema Linking

Schema linking is often used in the pipeline of Text-to-SQL systems in to filter-out unnecessary schema elements or to emphasize the most relevant elements, given a specific NL Question. Earlier works [6, 7, 18, 46] mostly relied on n-gram matching techniques to find mentions of DB elements in the NLQ. This allowed for specific parts of the NL Query and DB tables and columns to be specifically linked between each other, producing an additional input signal for the neural network. A different work [25] proposed a learned MLP classifier to predict such links, demonstrating that even a relatively simple schema linking model can improve performance in the text-to-SQL task. Although schema linking was proven beneficial, these approaches had their shortcomings. The string matching techniques are not robust to synonyms as they require that the question and schema follow the same phrasing and vocabulary. Furthermore, creating training data for a learned classifier can not be done automatically and requires a lot of manual work that is difficult to scale. Additionally, schema elements can sometimes be implied and not mentioned explicitly in the NL Query, making it impossible to identify them with this approach. For these reasons, more recent works have focused on only predicting the required tables and columns and not the part of the NLQ that mentions them [15]. They also rely on trained or pre-trained models that can generalize to harder cases where synonyms are used or some DB elements are implied. For these types of approaches, we observe to major categories based on the way the predictions are generated.

[1] https://github.com/IBM/few-shot-schema-linking

Ranking-based Schema Linking. On one hand we have ranking approaches where each schema element is assigned a relevance score, based on the given NL Query. Such an approach is proposed by RESDSQL [26] and later adopted by CodeS [27], where an encoder-only Language Model (e.g., RoBERTA [29]) is fine-tuned to generate relevance scores. This approach is more explainable as the score of each element can be examined to better understand the model's performance, however it requires to be fine-tuned and its thresholds must be tuned based on a specific dataset making it more difficult to transfer to a new dataset. To avoid setting thresholds, RESDSQL [26] proposes to keep the top-m tables and top-n columns based on the relevance scores, however this also requires setting the m and n parameters and can lead to low recall as we show in our experimental section.

Preliminary SQL Generation. On the other hand, another work [50] proposes to generate a preliminary SQL query with a Text-to-SQL model and extract the used columns and tables as schema linking predictions. On one hand this approach can perform schema linking without the need for an additional model, because it use the already available Text-to-SQL model. On the other hand, it is very likely that a preliminary SQL will contain a smaller amount of schema elements than the predictions of other systems, because they must all appear in a single query. As such it is more likely that such an approach might miss tables and columns that could have otherwise been predicted, leading to a lower recall which can be catastrophic. Additionally, if a model was able to generate a query that uses all the right schema elements, then it probable already good enough that it does not need schema linking.

LLM-based Schema Linking. Finally, there are approaches that rely on LLMs to predict the needed tables and columns as generated text. These approaches [12, 16, 35, 45, 49] are te most popular and use zero or few-shot prompting and ask the model to predict which tables and columns are required to generate the SQL that corresponds to a given NL Query. The advantage of these approaches is that LLMs can be used without the need of fine-tuning, making them easier to transfer to a new DB, and do not require any knob tuning as they do not need to set any thresholds. Our work focuses on this category of schema linking, as we work towards formalizing and evaluating this process that is often skimmed over in most systems. In fact, even though the aforementioned systems are some of the best performing in the literature, they do not offer many insights into how schema linking helps their system or how it could be improved. This work explores the different possible choices in the schema linking pipeline and derives insights on how to optimize the performance of this component.

2.3 Question Decomposition

The task of Question Decomposition has been previously proposed [33] as an approach for handling difficult examples of problems where a solution is already available but not as robust. While earlier works [13, 34] focused on automatically generating corpora of decomposed questions to train decomposition models, more recent works [22, 54] rely on the power of LLMs which are able to generate decompositions without the need for any fine-tuning. In the context of Text-to-SQL, there have been a few works that have integrated such techniques. MAC-SQL [45] uses Question Decomposition to guide the model towards a step-by-step reasoning during SQL prediction. Another work [47] proposes a weakly supervised approach through question decomposition that generates synthetic data by breaking down complex NL questions into simpler ones. Finally, on slightly different principle, the approach outlined in [30] incorporates active user feedback to enhance decomposed SQL generation. In the context of this work, we investigate whether schema linking could also benefit from decomposing complex questions and separately handling easier sub-questions.

3 Problem Definition

3.1 Text-to-SQL

The Text-to-SQL problem seeks to bridge the gap between natural language and structured data, allowing users to query RDBs using human language. Given a natural language question, the goal is to generate a corresponding SQL query that can be executed against a specified database (DB) to retrieve the desired information.

More formally, given a sequence of tokens $X = (x_i)_{i=1}^{t}$ representing a natural language question (NLQ) over a database $\mathcal{D}$, the Text-to-SQL task is to generate or transform it into a sequence of tokens representing the corresponding SQL query $Q = (q_i)_{i=1}^{r}$, formulated with respect to $\mathcal{D}$.

Database representation. The database $\mathcal{D}$ consists of n tables $T_1, T_2, ..., T_n$, where each table T_i contains the columns $C_{i1}, C_{i2}, ..., C_{im_i}$. The value m_i represents the number of columns in table T_i. We also define $t_{i1}, t_{i2}, ..., t_{im_i}$ being the types associated to the columns, and $k_{i1}, k_{i2}, ..., k_{im_i}$, k_j where $k \in$ {none, primary, foreign} indicating whether the column has no special properties or is a primary or a foreign key.

A SQL query Q is considered a successful generation for the NLQ X if it is: (i) syntactically correctness and executable for the database $\mathcal{D}$, and (ii) aligned with the user's intent, leading to the requested results.

3.2 Schema Linking

The Schema Linking problem aims at identifying the relevant DB elements (i.e., tables and columns) that are needed to construct a SQL query, given a NL query. It can be represented as a function $\phi(\cdot)$ that given an NLQ X produces a set of tables $\mathcal{T} = \{T_k\}_{k=1}^{s}$ and the corresponding columns within the tables $C = \{C_j^t\}_{j=1}^{l}$, $t \in \mathcal{T}$.

What must also be noted is that recall is a key assessment metric for this task because if a schema linking prediction misses a table or column that is needed for the SQL, then it will be impossible for the Text-to-SQL model to generate the correct SQL query.

The schema linking approaches in the literature usually consider the optimal set of columns C and tables $\mathcal{T}$ to contain minimum number of the schema elements necessary for SQL to be correct and executable, without adding any additional information by including some other valid columns from the schema $\mathcal{D}$. This way the Text-to-SQL model is aimed to operate on the result of schema linking that only contains the necessary columns and tables. In addition to the traditional schema linking oracle that contains the exact needed set of columns and tables, we also consider enriched schema (see Section 4.6) that might contain additional columns for better table representation.

4 Methodology

In this section we describe all the methodology and techniques used in our experiments. More specifically, we provide an overview

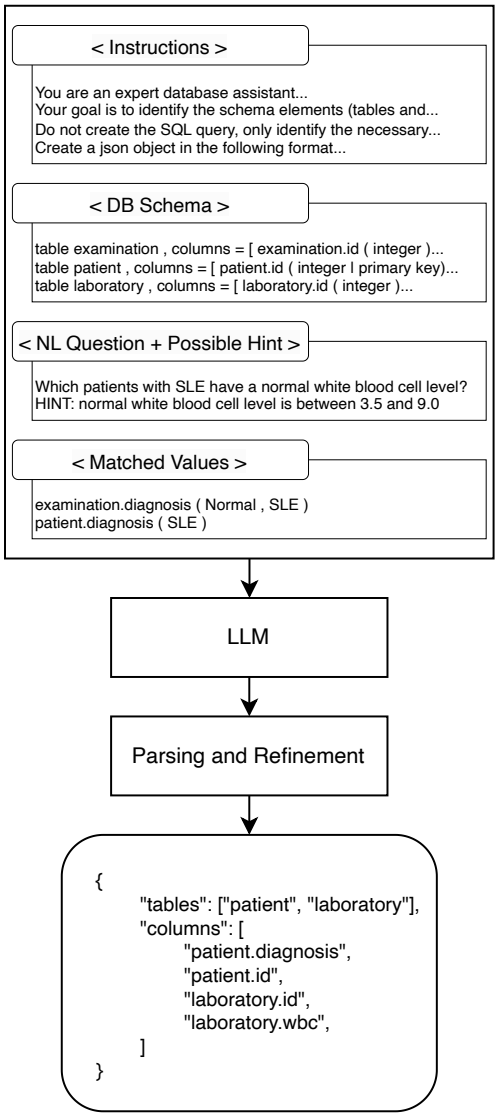


Figure 2: An overview of the schema linking pipeline.

of the pipeline used for in-context schema linking, the demonstration retrieval techniques, how question decomposition and fine-tuning are performed, the prediction refinement strategies, and how schema enrichment in used.

4.1 In-Context Schema Linking

To perform in-context (or few-shot) schema linking, we must define on one hand the structure of the given input and on the other hand the parsing of the generated output. We follow the findings presented in previous studies [9, 27] and construct our prompt using the following parts: (i) task instructions, (ii) DB schema, (iii) NL Question with additional hints when available, and (iv) matched values from the DB contents. Furthermore, in order to automatically extract the schema linking predictions from the LLM's textual output we chose a JSON output format that can easily be parsed.

Task Instructions refer to general guidelines (e.g., "You are an expert DB assistant") and task-specific instructions that explain how the model should behave. For example, we state that it should predict the needed tables and column and return a JSON dictionary, giving a sample of the expected output. We also state that the model should not generate a SQL query which is a common mistake that LLMs make even when asked to only predict tables and columns, The *DB Schema* (i.e., the DB tables and the columns in each table) is an essential input which can be enriched by also providing the types of the columns (e.g., text, date, number, etc.), the primary and foreign key properties of the columns, and exemplar values stored in each column. Additionally, some datasets provide more descriptive names for some DB elements that can help better understand their purpose. The *NL Question* is another essential input which can also be enriched in some cases where we might also have access to domain-specific evidence that help disambiguate the NLQ. For example, given the question *"List all patients with normal white blood cell level"*, a relevant evidence is that *"normal white blood cell level refers to WBC between 3.5 and 9.0"*. Finally, we add *Matched Values* from the DB that appear in the NL Query (e.g., the term "SLE" is found in the column "patient.diagnosis") which can be a strong indicator that the column in which the value appear is needed to build the SQL. To do this efficiently we use an approach from the CodeS [27] paper, where a BM25 index is built for the values of the DB to enable the fast retrieval of relevant values based on the NLQ. The retrieved values are then compared to the NLQ using Longest Common Sub-sequence (LCS) and the most relevant are kept.

Using this approach we can prompt any pre-trained LLM to generate schema linking predictions, allowing us to uniformly compare the performance of different models. An example of our pipeline can be seen in Figure 2. Finally, the predictions can then be used by any Text-to-SQL system, independently of its architecture, by filtering out all irrelevant schema elements and only presenting the predicted tables and columns.

4.2 Demonstration Retrieval for ICL

When performing In-Context Learning it is important to select demonstrations examples that are similar to the question presented to the LLM [16]. To retrieve similar examples we follow a structural similarity approach [27], which uses the semantic similarity and the structural similarity between the test question and the questions in the demonstration set. More specifically, for each test question we calculate two similarity scores with the demonstration questions: (i) sentence similarity using the original questions, and (ii) sentence similarity after removing all entities in the questions. The maximum between the two scores is then used as the similarity criterion. This approach helps retrieve questions that are similar in structure, which would have been otherwise overlooked because of the different entities that appear in them. Within the scope of this paper, we do not provide an analysis of the question similarities. An in-depth study by Pourreza et al. can be found in [36].

4.3 Prediction Refinement Techniques

While pre-trained LLMs show very good performance in understanding NL and identifying semantic relations between the NLQ and the DB elements, they often disregard the database logic that goes into building a SQL query. For example, they might not include the necessary foreign keys or a connecting table in their prediction, which are essential for the SQL query. Additionally, due to the generative nature of LLMs there is no constraint

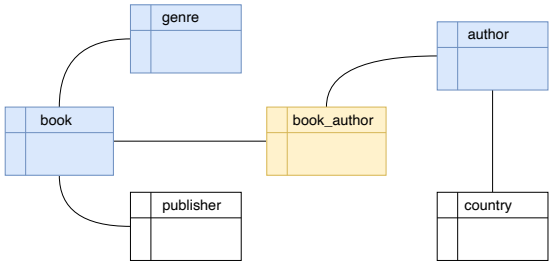


Figure 3: An example of foreign key-path refinement: The model predicts the blue tables, but the yellow table is also needed to generate the SQL query.

preventing the model from generating hallucinations or misspellings of tables and columns (e.g., "actor.birth_year" instead of "actor.birthYear"). Based on common mistakes, we investigate a set of automatic refinement techniques that can increase the recall of a model's predictions. Specifically we consider the following refinement steps:

A. Approximate String Matching: To tackle the hallucination and misspelling problem of LLMs we perform approximate string matching when a prediction does not exactly match a column or table name, to find the closest match. More specifically, for every unmatched table and column we calculate the approximate match scores with the equivalent DB tables or columns and keep the highest scoring candidate[2]. We only consider matches above a threshold of 0.5 so that hallucinations that do not correspond to any actual element do not get any assignments. If more than one DB elements achieve the highest similarity score, we include all tied elements, although this is rare. Additionally, for column names we calculate the approximation score with and without the table prefix and keep the average of the two scores. This is done to ensure that the predicted table has a weight in finding similar columns but does not overpower the actual column name. For example, consider the predicted column "airlines.date" that should be matched to "flights.date", but would mistakenly be matched to "airlines.name", because there is a bigger overlap between the table names even though only the "flights" tables has a column named "date".

B. Table Membership: If a column is predicted, we check that the table it belongs to is also present in the predictions. If the table is missing we add it automatically. While this might seem straight-forward there is no restriction preventing the model from omitting the table which can hinder the effectiveness of the following step.

C. Foreign Key Path: While LLMs are good at understanding semantic relationships between the question and the needed DB elements, they often disregard the relational model that connects the DB elements. For this reason, junction tables that are needed to connect tables used in the query are often overlooked. To tackle this we use a *Foreign Key Path* refinement which requires that all predicted tables must have a connecting path of foreign keys. An example can be seen in Figure 3 where the tables "author", "book", and "genre" have been predicted by the model, but can not be used in a SQL query unless the table "book_author", which connects authors and books in a many-to-many relationship, is also added.

[2]We implement this using the RapidFuzz (https://github.com/rapidfuzz/RapidFuzz) open source library for fast fuzzy string matching.

Example NLQ: "Among the countries whose GDP is over 1,000,000, how many of them have a population growth rate of over 3%?"
Example Decomposition:
- "What is the GDP of each country?"
- "What is the population growth rate of each country?"
- "Which countries have a GDP over 1,000,000?"
- "How many countries have a population growth rate of over 3%?"

Figure 4: An example of a decomposition for a Natural Language Question (NLQ).

D. Foreign Key (FK) Connection: Similarly to the previous refinement, we noticed that LLMs often miss important foreign key columns without which it is not possible to JOIN the predicted tables. To address this, if two predicted tables are connected by Foreign Keys, we make sure that the FK columns are also added to the predictions.

4.4 Question Decomposition

As previous works [45, 47] have shown, NLQs can refer to multiple DB elements in a very short span of text which can be confusing to the LLM, leading it to miss important tables or columns. In order to alleviate this challenge and help the model focus on all the referenced elements independently, we explore the impact of question decomposition for schema linking. This way schema linking can be performed on each question separately and the results of each prediction can be combined to obtain more informed schema links. An example of a decomposition is shown in Figure 4.

To generate the decomposition of the original question we follow the few-shot paradigm used in a previous Text-to-SQL system [45]. This is also similar to the few-shot approach of the schema linking pipeline, making the combination of the two approaches much simpler. The prompt contains instructions and a set of few-shot demonstrations that guide an LLM to follow specific requirements when decomposing the questions. Our requirements are that (i) each decomposed question can not be decomposed any further, (ii) there is no overlap or repetition between the decomposed questions, and (iii) no new information or comments are added by the LLM. More specifically, when question decomposition is used, schema linking is performed as follows. For each NLQ, we first use few-shot prompting to generate a set of decomposed questions. We then perform schema linking on each decomposed question separately to obtain its table and column predictions. The final schema links correspond to the union of each individual prediction.

4.5 Fine-tuning for Schema Linking

In addition to In-Context Learning for schema linking, we also experiment with Supervised Fine-Tuning (SFT) LLMs for the task. While this approach requires additional time and computational resources and is less portable, it is possible that a fine-tuned model will be able to achieve better performance than a more general one. Towards this direction we create a training corpus following a similar approach to the one we describe in Section 4.1 with the appropriate expected outputs in JSON format. In

this case we do not need to add the instructions that describe the expected behavior as this will be learned through the fine-tuning process. We present and compare the performance of fine-tuned LLMs in Section 6.

4.6 Schema Enrichment Investigation

In this section, we formulate the methods for schema enrichment that expand the linked schema beyond the strictly necessary columns for SQL generation. The standard 'oracle' schema ($\mathcal{T}_{base}$) contains the minimal columns for a specific query. However, LLMs might benefit from richer context about a table's overall structure, semantics, and content to reason effectively, especially for complex queries involving relationships or implicit concepts. We hypothesize that providing a more representative subset of columns, beyond the bare minimum, could improve the LLM's understanding of the table and lead to better SQL generation. We investigate two strategies to select these additional columns based on principles of diversity (covering different aspects of the table) and representativeness (including typical columns).

Example. Consider a table 'Products' with columns [ProductID(PK), Name, Category, Price, SupplierID(FK), Weight, Description]. If the oracle schema linking result for a query about product names is {ProductID, Name}, diversity enrichment might add 'Description' (semantically distant text) and representation enrichment might add 'Price' (typical numeric attribute) providing better understanding about the nature of the product entity. Such representation might help with the correct interpretation of a query's constraints and relationships.

1. Diversity. To ensure the diversity coverage of the resulting sub-schema, we measure the coverage span of $\mathcal{T}_{base}$ and identify elements in the initial schema $\mathcal{T}$ that are the most distant from the base set. We define the ***coverage span*** of a set of columns within a table as the extent to which these columns represent the different semantic concepts or data types present in the table, measured by the maximum distance between column embeddings within that set. Column embeddings are generated using a Sentence-BERT model [39] applied to a textual representation of each column, including its name, data type, table name, descriptive comments (if available), and sample data values. This captures the semantic meaning of the column in its database context.

For each table $T \in \mathcal{T}_{base}$, we consider the coverage span within the scope of that table. The columns of the base (oracle) schema are denoted as C_{in}, while the remaining columns are denoted as C_{out}. Within the column embedding space we compute two sets of distances D_{in} and D_{out}. D_{in} corresponds to the distances within the C_{in} set, i.e. for each column $c_{in} \in C_{in}$ we compute the minimum distance between the embedding of this column to any other column within C_{in} excluding itself:

$$d^i_{in} = min\{ED_{c \in C_{in} \setminus c^i_{in}}(embed(c), embed(c^i_{in}))\}$$

where ED stands for the Euclidean distance.

For the distances $D_{out} = \{d^i_{out}\}$ we compute the minimum distances between all the outside columns C_{out} to the insider columns within C_{in}, this way we check if there is a column that is not included in the result schema linking set which is quite different (far away) from the included columns. More formally:

$$d^i_{out} = min\{ED_{c \in C_{in}}(embed(c), embed(c^i_{out}))\}$$

In order to decide if any other column should be included in the enriched set, we find the maximum distances d^{max}_{in} and d^{max}_{out} within the D_{in} and D_{out} sets and compare them. If $d^{max}_{in} \geq d^{max}_{out}$ we consider that there is already enough diversity coverage in the schema linking set, otherwise we add the element with the maximum outside distance d^{max}_{out} and repeat the procedure until $d^{max}_{in} \geq d^{max}_{out}$. Essentially we start from a 'personalized' column set of schema linking, and expand towards a more diverse 'exploration' direction. Additionally, this approach avoids using arbitrary thresholds for the diversity of the enriched schema linking, so that it is purely based on the relative distances within the column embedding space.

2. Representation. We also complement diversity by enriching the general representation of the characteristic columns. For this purpose we consider two techniques to sample from the column embedding space. The first technique uses **uniform sampling** in the embedding space based on a predefined inclusion percentage parameter $\rho \in [0, 1]$. A value of $\rho = 0$ indicates no minimum requirement, while $\rho = 1$ includes all columns. Random sampling is still one of the most popular sampling techniques of representative subsample extraction [40]. The second proposed representation schema enrichment technique, **cluster-based sampling**, identifies clusters within the embedding space and selects the columns closest to the cluster centroids as representatives. The number of clusters is automatically determined based on a predefined element inclusion distance. This distance threshold specifies how far an element can be from a cluster to be included in it. The distance threshold can be set based on the distribution of embedding distances within and between tables.

5 Experimental Setup

5.1 Datasets

To evaluate our work we use the Spider and BIRD datasets, the two most widely used datasets in Text-to-SQL literature.

Spider [51] is a cross-domain dataset containing 10,000 NLQ-SQL pairs over 200 different DBs. It has been one of the main driving forces in the neural era of Text-to-SQL, providing a common framework to train and evaluate different models. The SQL queries of Spider contain a wide variety of SQL clauses such as `GROUP BY`, `HAVING`, `ORDER BY`, `LIMIT`, `JOIN`, `INTERSECT`, `EXCEPT`, `UNION`, `NOT IN`, `EXISTS`, `LIKE` as well as nested queries. The authors also provide a categorization of the examples based on 4 levels of difficulty: easy, medium, hard, and extra hard.

BIRD [28] is a dataset released in 2023 that follows the same structure as Spider but contains more complex SQL queries, larger databases, and NL Questions that require a deeper understanding of the DB and its domain. The SQL queries found in BIRD contain all the types of clauses used in Spider as well as CTEs (i.e., `WITH` statements), functions (e.g., STRFTIME, CAST, IIF, SUBSTR, etc.), and mathematical operations between columns. For the above reasons, BIRD is very challenging and is currently regarded as the standard benchmark for Text-to-SQL. For certain questions, the BIRD dataset also provides "evidences" that can be used to give some insights to the model. These evidences are often definitions. For example, to the question *"Rank schools by their average score in Writing where the score is greater than 499, showing their charter numbers."* the following evidence is given: ' *"Valid charter number means the number is not null"*. The authors of the dataset provide a difficulty categorization for the queries in the dev set based on 3 levels: Simple, Moderate, and Challenging.

Table 1 presents an overview and statistics of the used datasets. In order to provide a common difficulty axis for our experiments we merge Spider's "easy" and "medium" categories into the "simple" category of BIRD. This decision was made given the fact that

Dataset	Tables per DB (Min/Avg/Max)	Query Length (Min/Avg/Max)	JOINs per Query (Min/Avg/Max)	Query Difficulty			Total
				Simple	Moderate	Challenging	
Spider-train	2 / 5.25 / 26	18 / 109.64 / 577	0 / 0.54 / 8	4,775	1,758	2,126	8,659
Spider-dev	2 / 4.00 / 11	20 / 106.69 / 422	0 / 0.50 / 4	690	174	168	1,032
BIRD-train	2 / 7.56 / 65	23 / 170.10 / 804	0 / 0.77 / 6	N/A	N/A	N/A	9,428
BIRD-dev	3 / 6.81 / 13	29 / 160.01 / 1,446	0 / 0.91 / 6	925	465	144	1,534

Table 1: Statistics of the used datasets.

BIRD's "simple" queries are similar to Spider's "medium" (i.e., small number of JOINs and use of aggregation functions). The "hard' and "extra hard" difficulties from Spider are mapped to the "moderate" and "challenging" categories from BIRD respectively. Unfortunately BIRD provides a difficulty rating only for its dev set. By comparing the dataset statistics we observe that BIRD contains more tables per database, longer queries, and more joins per query. This combined with the additional SQL clauses and functions it uses, makes it a harder benchmark for schema linking and text-to-SQL.

For the schema linking task, we generate the ground truth labels for each dataset by extracting the tables and the columns that are used in each ground truth SQL query. It should be noted that some of these labels might not be entirely accurate all of the time, as there are cases where the same logic can be expressed with different SQL queries. While more or less the same tables and columns will be necessary despite the used syntax, there might still be minor differences (e.g., a COUNT() can be performed over an id column or the star operator and still return the same results).

5.2 Metrics

We employ three metrics to evaluate our experiments, the first two are used to evaluate the schema linking performance of a model, while the third metric evaluates the Text-to-SQL performance of model (using another model's schema linking predictions).

Schema Linking Precision is defined as the number of correctly predicted schema columns as a fraction of the total number of predicted columns. We calculate the average of this score across all examples in the dataset. This metric represents how many unnecessary columns (noise) are incorrectly added in a schema linking prediction, in relation to the number of predictions. A high score indicates a very precise predictions, while a low score indicates a noisy prediction.

Schema Linking Recall is defined as the percentage of examples in the dataset where all the necessary columns were predicted. Or equivalently: The percentage of examples having perfect recall. We employ a stricter variation of traditional recall given that for any given example, if schema linking recall is not perfect then Text-to-SQL is certain to fail. This score essentially describes the the upper bound on the percentage of examples where Text-to-SQL can be performed successfully.

Execution Accuracy: The percentage of predicted SQL queries that when executed return the same results as the ground truth SQL query. This has long been the most widely adopted metric for Text-to-SQL [51], despite its known short comings as it can produce false positives when the predicted SQL might return the same result even though its logic might not match the NL Question.

5.3 Models and Baselines

We will now provide an overview of the models and baselines we use in our experiments.

No/Oracle Schema Linking: No schema linking refers to using the entire DB schema and oracle schema linking refers to using the ground truth schema to simulate the hypothetical scenario of a model that can achieve perfect schema linking. We include these two baselines for two reasons: (i) to use "No Schema Linking" as a baseline for the lowest precision scores, and (ii) to examine the difference in Text-to-SQL execution accuracy compared to "No Schema Linking" an the possible gain that is left in comparison with "Oracle Schema Linking".

RESDSQL (RoBERTA-Large) (Baseline): This approach is taken directly from the RESDSQL [26] paper, using the RoBERTA [29] encoder-only LM with a classification head. The model is fine-tuned on the dataset it is used (i.e., Spider or BIRD) and can predict confidence scores for each table and column of the schema. The authors choose to keep the top-5 rated tables and the top-6 rated columns of each of these tables, as they achieved the best results. As we will also observe in the experimental results, this leads to high recall scores but low precision scores as more tables and columns are selected than necessary. This could be improved by making the number of top-k tables and columns dynamic, but is beyond the scope of this work.

Prel-SQL (Baseline): This approach follows a previously proposed method [50] to perform schema linking with a preliminary SQL query prediction. In this case, an LLM is asked to predict a SQL query as it would for Text-to-SQL but its prediction is only used to extract the tables and columns, which will be used as schema linking predictions. The intuition behind it is that the LLM might be able to use the correct DB elements but it might not use them in the correct manner.

LLMs: In our experiments we use multiple open-source LLMs to get a representative view on how different models perform. We focus on code-oriented models that are instruct-tuned so that the fit best for the tasks and the few-shot setting. We refrain from using very large models and prefer small and medium variants (i.e., up to 33B parameters), because the problem at hand is only a sub-task of the larger Text-to-SQL problem and dedicating too many resources would be prohibitive for many applications. Specifically we use: Codestral-22B [3], Deepseek-Coder-Instruct-(6.7B and 33B) [17], Llama-3-Instruct-8B [31], and CodeGemma-Instruct-7B [42].

Additionally, for question decomposition we use Llama-3-Instruct-70B in order to keep the presentation of the experiments simpler, as we noticed very good performance with little differences between other large generalist models we tried. More specifically, we performed an evaluation of 100 sampled predictions for both datasets to measure how often the decomposition

was missing any necessary information from the original question. We observed that 95% and 97% of decompositions retained all necessary information for Spider and BIRD accordingly.

6 Experimental Results

6.1 Schema Linking Performance

First of all we present our findings on the schema linking performance of different LLMs. Table 2 contains our results on the Spider and BIRD datasets. For each model, we consider four usage scenarios: (i) using the model in an few-shot setting (i.e., no fine-tuning), (ii) performing few-shot prompting with question decomposition, (iii) fine-tuning the model for the schema linking task, and (iv) fine-tuning the model and also using question decomposition.

	Spider		BIRD	
Schema Linking Model	**Precision**	**Recall**	**Precision**	**Recall**
Baselines				
No Schema Linking	15.01	100	10.09	100
RESDSQL [26]	18.56	99.70	21.19	88.07
Prel-SQL (Deepseek-Coder-33B)	76.59	91.48	81.67	59.51
Prel-SQL (Codestral-22B)	77.22	88.78	78.20	68.51
Few-Shot Prompting				
Deepseek-Coder-6.7B	**74.01**	90.61	63.11	73.07
Llama3-8B	68.36	78.33	63.46	59.06
Codestral-22B	72.37	97.29	**67.93**	**86.11**
Deepseek-Coder-33B	62.02	**97.48**	60.62	75.81
Few-Shot Prompting with Question Decomposition				
Deepseek-Coder-6.7B	55.85	**97.00**	**51.33**	85.78
Llama3-8B	51.52	93.52	49.39	80.70
Codestral-22B	**56.23**	96.51	45.35	**92.04**
Deepseek-Coder-33B	56.03	96.32	49.62	83.83
Supervised Fine-Tuning				
Deepseek-Coder-6.7B	**81.64**	93.32	**79.79**	**75.03**
Llama3-8B	67.04	74.95	69.25	55.86
Codestral-22B	74.16	86.55	74.75	54.62
Deepseek-Coder-33B	81.46	**93.13**	79.44	**75.03**
Supervised Fine-Tuning with Question Decomposition				
Deepseek-Coder-6.7B	**67.03**	96.42	66.02	**84.09**
Llama3-8B	54.49	88.39	59.29	68.44
Codestral-22B	59.77	93.03	63.68	68.70
Deepseek-Coder-33B	62.10	**96.51**	**66.23**	83.57

Table 2: Schema linking performance on the Spider and BIRD benchmarks. We consider different settings, using the LLM in either a few-shot paradigm or after fine-tuning it. We also experiment with question decomposition in both cases.

Regarding the general performance of schema linking, a recall of 100% means that all the tables and columns necessary to answer the question are present in the linking schema. A recall inferior to 100% is potentially detrimental as the model will not receive the minimum information to answer the question. A precision inferior to 100% means that the linked schema will contain additional tables or columns to those necessary.

Firstly, we observe that the baselines (RESDSQL and Prel-SQL) can achieve good performance in Spider but their performance degrades in the harder BIRD benchmark. RESDSQL achieves high recall scores but very low precision, which was expected given that it is designed to predict more elements than necessary most of the time. Prel-SQL approaches go in the opposite direction, achieving higher precision scores but lower recall because of their design, which predicts a much lower number of DB elements. However, in most cases, LLM-based approaches produce better results.

On the Spider benchmark, we observe that all LLMs can consistently achieve high recall and provide a precision that is four to five times higher than the baseline of "No Schema Linking", thus greatly reducing the noise of the not needed DB elements. The best recall is achieved by Deepseek-6.7B using supervised fine-tuning. However, even simple few-shot prompting can achieve very good performance with recall up to 97% and precision up to 74%. It is also clear that each model performs better in different usage scenarios. For example, Codestral-22B performs better with few-shot prompting, while the DeepSeek benefits from fine-tuning.

On the BIRD benchmark, models perform relatively worse, given the much higher complexity and difficulty of the benchmark, but still provide a large increase in precision compared to baselines with a recall that can reach above 80% and up to 92.04%. The best recall is achieved by Codestral-22B using few-shot prompting with question decomposition and the best precision is achieved by Deepseek-6.7B using supervised fine-tuning.

By comparing the performance of each model between the two benchmarks, we observe that bigger models do not always translate to better results. In fact, while larger models usually perform better, a small code-oriented LLM like DeepSeek-Coder-6.7B is quite competitive and in some cases the strongest model. Especially when comparing fine-tuned models, the playing ground is much more even, which is an indicator that we might not need huge models for schema linking, but specialist models instead.

> **Takeaway #1**: *Using larger models will usually lead to better results, but small specialist models (e.g., code-oriented, fine-tuned) are still competitive and sometimes even better for the schema linking task.*

Additionally, we make two observations across both datasets: (i) Question Decomposition usually increases schema linking recall, sometimes by more than 20%, while decreasing precision, and (ii) Supervised Fine-Tuning usually increases precision while decreasing recall. The first observation is expected, given that sampling additional outputs will increase the number of predicted elements that enables the model to find more useful elements at the cost of also adding noise. The second observation shows that fine-tuning a model will make it more cautious to the amount of elements it predicts, most likely because it tries to generate a similar number of elements to the ground truths it has been trained on. Additionally, we observe that the combination of both techniques leads to more balanced precision and recall scores, keeping the precision boost of fine-tuning while minimizing the decrease in recall.

We further investigate the effect of question decomposition by performing an analysis on the schema linking predictions of the Codestral-22B model with and without QD. We observe that in 517 examples recall is increased and in 71 examples recall is decreased, showing that decomposition is beneficial more often than it causes performance degradation. For example, in the BIRD question *"Among the superheroes with height from 170 to 190, list the names of the superheroes with no eye color."*, when not using QD, the model does not predict the required columns `colour.id, colour.colour`. However, the model manages to find these columns when the question is decomposed into the

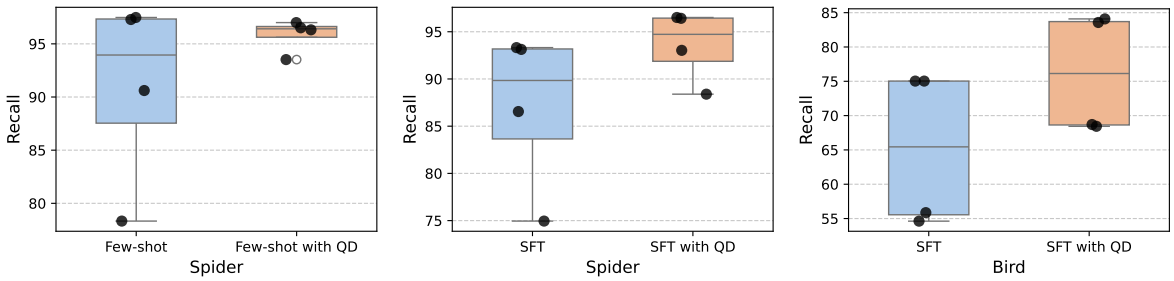


Figure 5: Comparison of schema linking recall scores for different scenarios with and without Question Decomposition

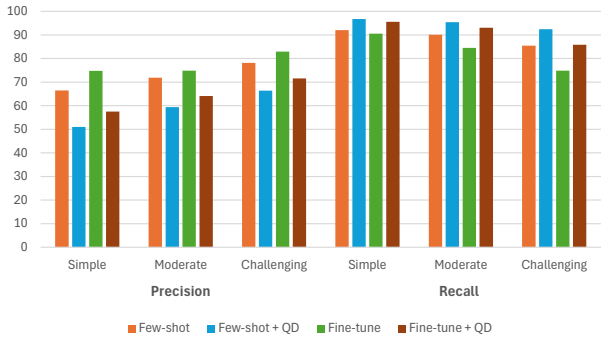
following sub-questions: *"What is the height of each superhero?"*, *"Which superheroes have a height between 170 and 190?"*, *"What is the eye color of each superhero?"*, *"Which superheroes have no eye color?"*. This is due to the fact that the model sees simpler questions and multiple references of "color", compared to the initial question that only mentions it once among many other attributes.

In contrast, recall degrades in the BIRD question *"State different accounts who have account opening date before 1997 and own an amount of money greater than 3000USD."*, where the model misses the column `trans.amount` when using QD. This happens because the word "amount" is not used in the decomposition: *"What is the account opening date of each account?"*, *"Which accounts have an account opening date before 1997?"*, *"What is the current balance of each account?"*, *"Which accounts have a balance greater than 3000USD?"*. The decomposition used the word "balance" instead, making it harder for the model to find the needed column.

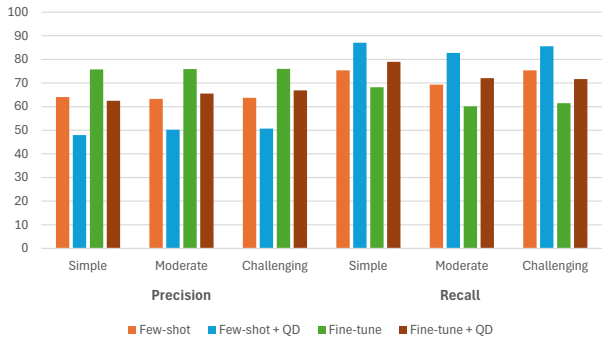
Our statistical analysis using t-tests and non-parametric Mann-Whitney U tests revealed a statistically significant differences between methods with and without QD in 5 out of 8 experimental settings (precision and recall for Spider and Bird, with and without SFT conditions). In the remaining three scenarios (Figure 5), usage of QD consistently resulted in higher recall values, but the effect did not reach statistical significance given the current number of observations.

> **Takeaway #2**: *Supervised Fine-tuning tends to increase precision and decrease recall, while Question Decomposition increases recall and decreases precision. A combination of both can be used for more balanced performance.*

Finally, we present a performance comparison of different approaches based on the query difficulty presented in Section 5.1. Figure 6 shows the average performance of each usage scenario of Table 2. This aggregation of results allows us to further pinpoint our previous observation on the trade-offs of question decomposition and fine-tuning. Question decomposition leads to higher recall and lower precision, whereas fine-tuning leads to higher precision and lower recall. We also observe a pattern when query difficulty increases, where precision increases and recall drops. This trend is stronger in the Spider dataset than in BIRD, where scores sometimes remain similar between difficulty levels. This could be attributed to the fact that harder queries contain more DB elements and as such models are more likely to miss more relevant elements.



(a) Results on Spider



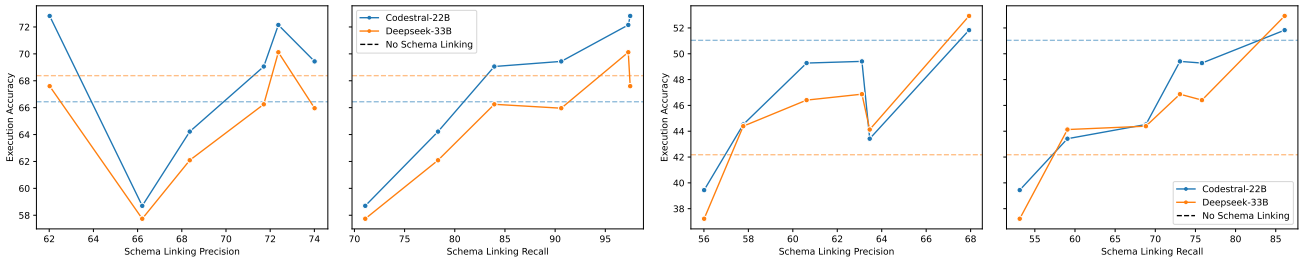
(b) Results on BIRD

Figure 6: Performance comparison on query difficulty of different LLM-based schema linking approaches.

6.2 The impact of schema linking on Text-to-SQL

Moving on, we investigate the effect of schema linking on the Text-to-SQL performance of LLMs. More specifically, for the Text-to-SQL task we use Codestral-22B and Deepseek-33B in a few-shot setting, and take the schema linking predictions of the few-shot models presented in Table 2. Our results can be seen in Figure 7.

We observe that across both datasets, despite a few outliers that we will discuss, better schema linking leads to better text-to-SQL generation. It is also apparent from the clearer upward trend in the figures, that recall plays a much more important role than precision, which is expected as we discussed above. Precision is also important, but it can be overshadowed by a low recall score, as the outliers in the precision figures show. Specifically, in Figure 7a the schema linking prediction with the lowest precision leads to very high execution accuracy as it corresponds to the



(a) Results on Spider

(b) Results on BIRD

Figure 7: Text-to-SQL performance with respect to the different schema linking scores of the tested LLMs. The horizontal "No Schema Linking" line shows the baseline Text-to-SQL performance of each model without any schema linking method.

schema links with the highest recall, while the second-lowest precision leads to poor execution accuracy because it corresponds to the schema links with the lowest recall. Similarly, in Figure 7b the second-highest precision shows a sudden dip in execution accuracy because it corresponds to the schema links with the second-lowest recall.

Another interesting observation can be made when comparing Text-to-SQL performance with and without schema linking (horizontal dashed lines). It becomes apparent that schema linking might not always improve execution accuracy, especially when the Text-to-SQL model can already achieve good performance without it. For example, Codestral-22B has a much higher execution accuracy on BIRD without schema linking compared to Deepseek-33B, and only the best schema linking performance can lead to an improvement on Text-to-SQL. On the other hand, Deepseek-33B benefits by almost all schema linking models as it starts from a lower point.

> **Takeaway #3**: *Better schema linking performance leads to better Text-to-SQL performance with recall having a more important role than precision. However, not all Text-to-SQL models benefit from schema linking to the same extent; the better the model is, the less additional schema linking it needs.*

6.3 The Impact of Schema Enrichment

We evaluated the effectiveness of the schema enrichment techniques introduced in Section 4.6, first applied to the Oracle schema linking performance and then to our schema linking approach.

For these experiments, we consider the proposed diversity - representativeness enrichment presented in Section 4.6. We consider both proposed approaches: *sampling* and *clustering*. The sampling approach makes use of the 'all-MiniLM-L12-v2' sentence transformer embedding [1] to compute contextual column embeddings. This approach is parametrized by the inclusion percentage. The clustering approach uses the DBSCAN algorithm [14] to automatically identify density-based clusters in the high-dimensional column embedding space. Also, we consider the alternative whether outliers are included or not. The experimental results are summarized in Table 3 and graphically depicted in Figures 8 and 9.

First, we check our hypothesis that optimal schema linking lies beyond the schema linking oracle. Given the oracle extension results represented in the upper part of Table 3 and also in Figure 8, we can see that in two out of three tried LLMs for SQL generation indeed can go beyond oracle. Deepseek-33B gets a boost of over 5 percentage points while CodeGemma-7B gets slightly

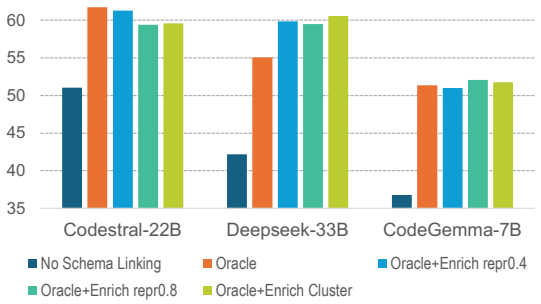


Figure 8: Exploration of the accuracy of Text-to-SQL beyond standard Oracle

better with maximum improvement of 0.72 points. Codestral-22B does not benefit much from the enriched schema going beyond oracle, but also does not degrade showing neutral changes. At the same time, when enrichment is used on the automatically extracted schema linking (lower part of Table 3 and Figure 9), Codestral-22B benefits by more than 2.5 percentage points, similar to CodeGemma-7B. Conversely, Deepseek-33b performed best with the original schema linking predictions and showed slight degradation with enrichment. This may indicate that the basis for enrichment was not as effective for this model, despite it showing the most potential for improvement over the oracle schema.

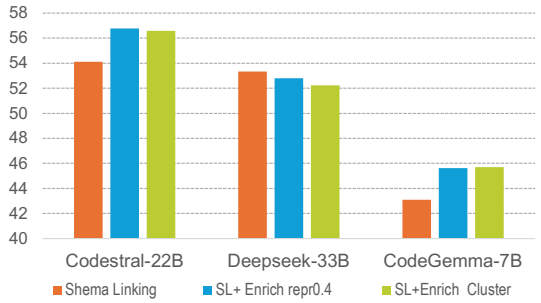


Figure 9: Text-to-SQL execution accuracy results for different versions of schema linking.

Schema Linking Model	Schema Linking		Text-to-SQL Execution Accuracy with		
	Precision	Recall	Codestral-22B	Deepseek-33B	CodeGemma-7B
Baselines					
No Schema Linking	10.09	100	51.04	42.18	36.77
Oracle Schema Linking	100	100	**61.73**	55.08	51.37
Oracle Enrichment					
Enriched Oracle (repr=0.3)	82.66	100	58.74	**60.76**	51.69
Enriched Oracle (repr=0.4)	84.04	100	61.28	59.84	50.98
Enriched Oracle (repr=0.6)	87.98	100	60.37	58.93	50.26
Enriched Oracle (repr=0.8)	92.93	100	59.39	59.52	**52.09**
Enriched Oracle (Cluster)	87.27	100	59.71	59.84	50.65
Enriched Oracle (Cluster + Outliers)	95.36	100	59.58	60.56	51.75
Prediction Enrichment					
Deepseek-6.7B (Finetuned)	79.79	75.03	54.11	**53.32**	43.09
DeepSeek-6.7B (Finetuned) + Enriched (repr=0.4)	67.71	75.94	**56.78**	52.80	45.63
DeepSeek-6.7B (Finetuned) + Enriched (Cluster)	70.44	75.55	56.58	52.22	**45.70**

Table 3: Experiments with schema enrichment. The middle part shows experiments on enriching the oracle (ground truth) schema links; bold numbers indicate best performance between oracle and enrichment. The bottom parts shows experiments on enriching the predictions of a schema linking model; bold numbers indicate best performance between the initial and enriched predictions.

> **Takeaway #4**: *Schema enrichment can improve both the predicted and oracle schema links, leading to higher Text-to-SQL scored for certain models. The "oracle" schema linking is in fact not always the optimal for Text-to-SQL generation.*

7 Ablation Results

7.1 The Impact of Prediction Refinement

We evaluate the impact of our proposed refinement techniques (Section 4.3) on the schema linking performance of LLMs. As we have already noted, these techniques aim to counter the lack of DB intelligence observed by LLMs (e.g., missing foreign keys, connecting tables, etc.). Table 4 shows the impact of these techniques averaged over all the experiments conducted in our study.

	Precision	Recall
Avg. Score Without Refinement	65.90	43.07
Avg. Score With Refinement	61.15	69.40
Ablation of Refinement Steps		
No Fuzzy Matching	61.39	66.54
No Parent Table	61.28	67.89
No FK Paths	61.88	67.70
No FK Connections	65.68	44.10

Table 4: Overview of average schema linking performance with and without prediction refinement, across all experiments on the BIRD dataset. We also perform a leave-one-out ablation of each refinement step.

The proposed techniques increase the recall achieving an average boost of 26.33% while reducing precision to much smaller extent, by 4.75% on average. The decreased precision is an acceptable sacrifice, since without the refinement about 30% of examples would not be converted to SQL at all. Furthermore, our ablation results show that the 'FK Connections' refinement technique has the largest impact among the techniques used. In fact, when removing the step the boost in recall is reduced by the largest amount, while precision stays almost the same. This supports the initial intuition that LLMs often disregard the foreign key columns that are necessary to build the desired SQL query.

> **Takeaway #5**: *The schema linking recall of LLMs can greatly be improved by expert-made rules that convey essential database knowledge.*

7.2 The impact of demonstration examples

Providing additional few-shot demonstration examples can be beneficial as it gives the model a better understanding of the task and the format in which it should respond. However, it is important to investigate how many examples are actually beneficial and exactly what their impact is on the tasks' metrics. Moving on, we investigate the impact of the number of demonstration examples and report precision and recall with respect to the number of demonstrations. Our results can be seen in Figure 10.

From our results we can make a few interesting observations. First of all we see that additional observations lead to an increase in precision, but after a certain amount of demonstrations (around 3-5) this effect seems to reach a plateau, or even degrade. On the other hand, recall appears to be less affected, fluctuating near the same score for most models. The Codestral model benefits greatly from having at least 1 demonstration, showing a great increase in both precision recall, but adding more than 1 demonstration has a relatively small effect. This indicates that this model is having trouble following the task and the requested output format without seeing a demonstration in its input. The Llama model improves precision by having up to three demonstrations but its performance starts to degrade when adding more. This is most

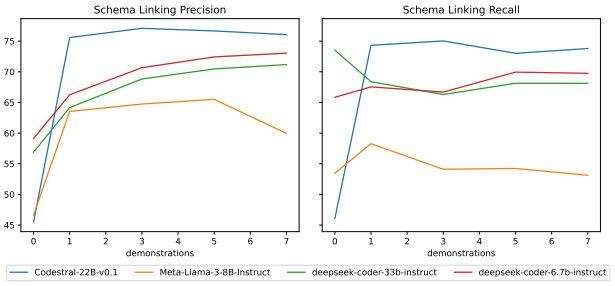


Figure 10: Schema linking performance on the BIRD dataset w.r.t the number of few-shot demonstrations.

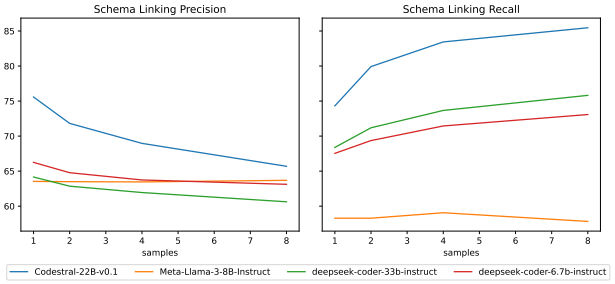


Figure 11: Schema linking performance on the BIRD dataset w.r.t the number of sampled outputs.

likely an indication that the model struggles to process such a large input.

> **Takeaway #6**: *Having at least one demonstration example can be very beneficial as it benefits the model's understanding of the task. However adding more demonstrations should be done with caution based on each model's behaviour.*

7.3 The impact of additional output sampling

LLMs give us the opportunity to sample multiple schema linking predictions for each input and use their union as the final prediction. The intuition is that the prediction with the highest confidence with might miss some DB tables and columns that could have been predicted by using the second or third highest-ranked predictions. To evaluate how the number of sampled outputs affects schema linking performance, we keep the number of few-shot demonstrations constant at 1 and run inference with different numbers of sampled predictions. Our results can be seen in Figure 11.

Our experiments show a trend across most models: increasing the number of samples leads to decreased precision and increased recall. This also confirms our intuition that considering more predictions for each input allows the model to generate needed DB elements that it would have otherwise missed. This comes at the cost of also generating tables and columns that are not useful, leading to a decrease in precision. However, given the importance of recall for the problem we argue that it is beneficial to explore this direction.

> **Takeaway #7**: *Sampling additional outputs from LLMs helps retrieve otherwise missed DB elements, increasing schema linking recall but decreasing precision.*

8 Conclusion

Our study presents an in-depth analysis of LLM-based schema linking under different usage scenarios and their effect on Text-to-SQL. Using a unified pipeline allowed us to apply popular techniques from the Text-to-SQL field, such as question decomposition and supervised fine-tuning, along with a set of refinement techniques, that can boost the performance of LLMs. Our experimental findings reveal several takeaways for improving and effectively incorporating schema linking in a Text-to-SQL pipeline. One main conclusion is that schema linking in low resource settings is possible and can lead to better SQL generation when used correctly. We also challenged the assumption that the oracle schema, which contains only the minimum necessary columns for SQL generation, is always the best schema representation. In conclusion, our work provides valuable insights and methodologies for enhancing Text-to-SQL systems through effective schema linking.

Artifacts

The code that was developed for this work and used for all the presented experiments is made publicly available at https://github.com/IBM/few-shot-schema-linking.

References

[1] [n. d.]. Sentence Transformer model: all-MiniLM-L12-v2. https://huggingface.co/sentence-transformers/all-MiniLM-L12-v2

[2] Katrin Affolter, Kurt Stockinger, and Abraham Bernstein. 2019. A comparative survey of recent natural language interfaces for databases. *The VLDB Journal* 28, 5 (Oct. 2019), 793–819. doi:10.1007/s00778-019-00567-8

[3] Mistral AI. [n. d.]. Codestral: Hello, World! https://mistral.ai/news/codestral/. Accessed: 2024-08-07.

[4] Sihem Amer-Yahia, Jasmina Bogojeska, Roberta Facchinetti, Valeria Franceschi, Aristides Gionis, Katja Hose, Georgia Koutrika, Roger Kouyos, Matteo Lissandrini, Silviu Maniu, Katsiaryna Mirylenka, et al. 2025. Towards Reliable Conversational Data Analytics. In *28th International Conference on Extending Database Technology (EDBT), Barcelona, Spain, 25-28 March 2025*, Vol. 28. Open Proceedings, 962–969.

[5] I. Androutsopoulos, G.D. Ritchie, and P. Thanisch. 1995. Natural language interfaces to databases – an introduction. *Natural Language Engineering* 1, 1 (1995), 29–81. doi:10.1017/S135132490000005X

[6] Ben Bogin, Jonathan Berant, and Matt Gardner. 2019. Representing Schema Structure with Graph Neural Networks for Text-to-SQL Parsing. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4560–4565. doi:10.18653/v1/P19-1448

[7] Ursin Brunner and Kurt Stockinger. 2021. ValueNet: A Natural Language-to-SQL System that Learns from Database Information. arXiv:2006.00888 [cs.DB] https://arxiv.org/abs/2006.00888

[8] Robin Chan, Katsiaryna Mirylenka, Thomas Gschwind, Christoph Miksovic, Paolo Scotton, Enrico Toniato, and Abdel Labbi. 2024. Adapting LLMs for Structured Natural Language API Integration. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing: Industry Track*. 991–1000.

[9] Shuaichen Chang and Eric Fosler-Lussier. 2023. How to Prompt LLMs for Text-to-SQL: A Study in Zero-shot, Single-domain, and Cross-domain Settings. arXiv:2305.11853 [cs.CL] https://arxiv.org/abs/2305.11853

[10] E. F. Codd. 1974. Seven Steps to Rendezvous with the Casual User. In *IFIP Working Conference Data Base Management*. https://api.semanticscholar.org/CorpusID:28690513

[11] Naihao Deng, Yulong Chen, and Yue Zhang. 2022. Recent Advances in Text-to-SQL: A Survey of What We Have and What We Expect. In *Proceedings of the 29th International Conference on Computational Linguistics*, Nicoletta Calzolari, Chu-Ren Huang, Hansaem Kim, James Pustejovsky, Leo Wanner, Key-Sun Choi, Pum-Mo Ryu, Hsin-Hsi Chen, Lucia Donatelli, Heng Ji, Sadao Kurohashi, Patrizia Paggio, Nianwen Xue, Seokhwan Kim, Younggyun Hahm, Zhong He, Tony Kyungil Lee, Enrico Santus, Francis Bond, and Seung-Hoon Na (Eds.). International Committee on Computational Linguistics, Gyeongju, Republic of Korea, 2166–2187. https://aclanthology.org/2022.coling-1.190/

[12] Xuemei Dong, Chao Zhang, Yuhang Ge, Yuren Mao, Yunjun Gao, lu Chen, Jinshu Lin, and Dongfang Lou. 2023. C3: Zero-shot Text-to-SQL with ChatGPT. arXiv:2307.07306 [cs.CL] https://arxiv.org/abs/2307.07306

[13] Dheeru Dua, Shivanshu Gupta, Sameer Singh, and Matt Gardner. 2022. Successive Prompting for Decomposing Complex Questions. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*,

Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 1251–1265. doi:10.18653/v1/2022.emnlp-main.81

[14] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. 1996. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd*, Vol. 96. 226–231.

[15] Ju Fan, Zihui Gu, Songyue Zhang, Yuxin Zhang, Zui Chen, Lei Cao, Guoliang Li, Samuel Madden, Xiaoyong Du, and Nan Tang. 2024. Combining Small Language Models and Large Language Models for Zero-Shot NL2SQL. *Proceedings of the VLDB Endowment* 17, 11 (2024), 2750–2763.

[16] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2024. Text-to-SQL Empowered by Large Language Models: A Benchmark Evaluation. *Proc. VLDB Endow.* 17, 5 (may 2024), 1132–1145. doi:10.14778/3641204.3641221

[17] Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Y. Wu, Y. K. Li, Fuli Luo, Yingfei Xiong, and Wenfeng Liang. 2024. DeepSeek-Coder: When the Large Language Model Meets Programming – The Rise of Code Intelligence. arXiv:2401.14196 [cs.SE] https://arxiv.org/abs/2401.14196

[18] Jiaqi Guo, Zecheng Zhan, Yan Gao, Yan Xiao, Jian-Guang Lou, Ting Liu, and Dongmei Zhang. 2019. Towards Complex Text-to-SQL in Cross-Domain Database with Intermediate Representation. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, Anna Korhonen, David Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, Florence, Italy, 4524–4535. doi:10.18653/v1/P19-1444

[19] Noah Hampp and Katya Mirylenka. 2024. Leveraging Large Language Models for Natural Language to SQL Conversion (Reward Modeling and RLAIF for Improved Natural Language to SQL Generation). In *The Mathematics of Machine Learning Workshop, ETH Zurich.*

[20] Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. 2025. Next-Generation Database Interfaces: A Survey of LLM-based Text-to-SQL. arXiv:2406.08426 [cs.CL] https://arxiv.org/abs/2406.08426

[21] George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A Survey on Deep Learning Approaches for Text-to-SQL. *The VLDB Journal* (2023). doi:10.1007/s00778-022-00776-8

[22] Tushar Khot, Harsh Trivedi, Matthew Finlayson, Yao Fu, Kyle Richardson, Peter Clark, and Ashish Sabharwal. 2023. Decomposed Prompting: A Modular Approach for Solving Complex Tasks. In *The Eleventh International Conference on Learning Representations*. https://openreview.net/forum?id=_nGgzQjzaRy

[23] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: Where are we today? *Proceedings of the VLDB Endowment* 13, 10 (2020), 1737–1750.

[24] Hyeonji Kim, Byeong-Hoon So, Wook-Shin Han, and Hongrae Lee. 2020. Natural language to SQL: where are we today? *Proc. VLDB Endow.* 13, 10 (June 2020), 1737–1750. doi:10.14778/3401960.3401970

[25] Wenqiang Lei, Weixin Wang, Zhixin Ma, Tian Gan, Wei Lu, Min-Yen Kan, and Tat-Seng Chua. 2020. Re-examining the Role of Schema Linking in Text-to-SQL. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 6943–6954. doi:10.18653/v1/2020.emnlp-main.564

[26] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling Schema Linking and Skeleton Parsing for Text-to-SQL. *Proceedings of the AAAI Conference on Artificial Intelligence* 37, 11 (Jun. 2023), 13067–13075. doi:10.1609/aaai.v37i11.26535

[27] Haoyang Li, Jing Zhang, Hanbing Liu, Ju Fan, Xiaokang Zhang, Jun Zhu, Renjie Wei, Hongyan Pan, Cuiping Li, and Hong Chen. 2024. CodeS: Towards Building Open-source Language Models for Text-to-SQL. *Proc. ACM Manag. Data* 2, 3, Article 127 (may 2024), 28 pages. doi:10.1145/3654930

[28] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, Xuanhe Zhou, Ma Chenhao, Guoliang Li, Kevin Chang, Fei Huang, Reynold Cheng, and Yongbin Li. 2023. Can LLM Already Serve as A Database Interface? A BIg Bench for Large-Scale Database Grounded Text-to-SQLs. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 42330–42357. https://proceedings.neurips.cc/paper_files/paper/2023/file/83fc8fab1710363050bbd1d4b8cc0021-Paper-Datasets_and_Benchmarks.pdf

[29] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. RoBERTa: A Robustly Optimized BERT Pretraining Approach. arXiv:1907.11692 [cs.CL]

[30] Antonis Mandamadiotis, Georgia Koutrika, and Sihem Amer-Yahia. 2024. Guided SQL-Based Data Exploration with User Feedback. In *2024 IEEE 40th International Conference on Data Engineering (ICDE)*. 4884–4896.

[31] AI Meta et al. 2024. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783* 2 (2024).

[32] OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, Irwan Bello, Jake Berdine, Gabriel Bernadett-Shapiro, Christopher Berner, Lenny Bogdonoff, Oleg Boiko, Madelaine Boyd, Anna-Luisa Brakman, Greg Brockman, Tim Brooks, Miles Brundage, Kevin Button, Trevor Cai, Rosie Campbell, Andrew Cann, Brittany Carey, Chelsea Carlson, Rory Carmichael, Brooke Chan, Che Chang, Fotis Chantzis, Derek Chen, Sully Chen, Ruby Chen, Jason Chen, Mark Chen, Ben Chess, Chester Cho, Casey Chu, Hyung Won Chung, Dave Cummings, Jeremiah Currier, Yunxing Dai, Cory Decareaux, Thomas Degry, Noah Deutsch, Damien Deville, Arka Dhar, David Dohan, Steve Dowling, Sheila Dunning, Adrien Ecoffet, Atty Eleti, Tyna Eloundou, David Farhi, Liam Fedus, Niko Felix, Simón Posada Fishman, Juston Forte, Isabella Fulford, Leo Gao, Elie Georges, Christian Gibson, Vik Goel, Tarun Gogineni, Gabriel Goh, Rapha Gontijo-Lopes, Jonathan Gordon, Morgan Grafstein, Scott Gray, Ryan Greene, Joshua Gross, Shixiang Shane Gu, Yufei Guo, Chris Hallacy, Jesse Han, Jeff Harris, Yuchen He, Mike Heaton, Johannes Heidecke, Chris Hesse, Alan Hickey, Wade Hickey, Peter Hoeschele, Brandon Houghton, Kenny Hsu, Shengli Hu, Xin Hu, Joost Huizinga, Shantanu Jain, Shawn Jain, Joanne Jang, Angela Jiang, Roger Jiang, Haozhun Jin, Denny Jin, Shino Jomoto, Billie Jonn, Heewoo Jun, Tomer Kaftan, Łukasz Kaiser, Ali Kamali, Ingmar Kanitscheider, Nitish Shirish Keskar, Tabarak Khan, Logan Kilpatrick, Jong Wook Kim, Christina Kim, Yongjik Kim, Jan Hendrik Kirchner, Jamie Kiros, Matt Knight, Daniel Kokotajlo, Łukasz Kondraciuk, Andrew Kondrich, Aris Konstantinidis, Kyle Kosic, Gretchen Krueger, Vishal Kuo, Michael Lampe, Ikai Lan, Teddy Lee, Jan Leike, Jade Leung, Daniel Levy, Chak Ming Li, Rachel Lim, Molly Lin, Stephanie Lin, Mateusz Litwin, Theresa Lopez, Ryan Lowe, Patricia Lue, Anna Makanju, Kim Malfacini, Sam Manning, Todor Markov, Yaniv Markovski, Bianca Martin, Katie Mayer, Andrew Mayne, Bob McGrew, Scott Mayer McKinney, Christine McLeavey, Paul McMillan, Jake McNeil, David Medina, Aalok Mehta, Jacob Menick, Luke Metz, Andrey Mishchenko, Pamela Mishkin, Vinnie Monaco, Evan Morikawa, Daniel Mossing, Tong Mu, Mira Murati, Oleg Murk, David Mély, Ashvin Nair, Reiichiro Nakano, Rajeev Nayak, Arvind Neelakantan, Richard Ngo, Hyeonwoo Noh, Long Ouyang, Cullen O'Keefe, Jakub Pachocki, Alex Paino, Joe Palermo, Ashley Pantuliano, Giambattista Parascandolo, Joel Parish, Emy Parparita, Alex Passos, Mikhail Pavlov, Andrew Peng, Adam Perelman, Filipe de Avila Belbute Peres, Michael Petrov, Henrique Ponde de Oliveira Pinto, Michael, Pokorny, Michelle Pokrass, Vitchyr H. Pong, Tolly Powell, Alethea Power, Boris Power, Elizabeth Proehl, Raul Puri, Alec Radford, Jack Rae, Aditya Ramesh, Cameron Raymond, Francis Real, Kendra Rimbach, Carl Ross, Bob Rotsted, Henri Roussez, Nick Ryder, Mario Saltarelli, Ted Sanders, Shibani Santurkar, Girish Sastry, Heather Schmidt, David Schnurr, John Schulman, Daniel Selsam, Kyla Sheppard, Toki Sherbakov, Jessica Shieh, Sarah Shoker, Pranav Shyam, Szymon Sidor, Eric Sigler, Maddie Simens, Jordan Sitkin, Katarina Slama, Ian Sohl, Benjamin Sokolowsky, Yang Song, Natalie Staudacher, Felipe Petroski Such, Natalie Summers, Ilya Sutskever, Jie Tang, Nikolas Tezak, Madeleine B. Thompson, Phil Tillet, Amin Tootoonchian, Elizabeth Tseng, Preston Tuggle, Nick Turley, Jerry Tworek, Juan Felipe Cerón Uribe, Andrea Vallone, Arun Vijayvergiya, Chelsea Voss, Carroll Wainwright, Justin Jay Wang, Alvin Wang, Ben Wang, Jonathan Ward, Jason Wei, CJ Weinmann, Akila Welihinda, Peter Welinder, Jiayi Weng, Lilian Weng, Matt Wiethoff, Dave Willner, Clemens Winter, Samuel Wolrich, Hannah Wong, Lauren Workman, Sherwin Wu, Jeff Wu, Michael Wu, Kai Xiao, Tao Xu, Sarah Yoo, Kevin Yu, Qiming Yuan, Wojciech Zaremba, Rowan Zellers, Chong Zhang, Marvin Zhang, Shengjia Zhao, Tianhao Zheng, Juntang Zhuang, William Zhuk, and Barret Zoph. 2024. GPT-4 Technical Report. arXiv:2303.08774 [cs.CL] https://arxiv.org/abs/2303.08774

[33] Pruthvi Patel, Swaroop Mishra, Mihir Parmar, and Chitta Baral. 2022. Is a Question Decomposition Unit All We Need?. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing*, Yoav Goldberg, Zornitsa Kozareva, and Yue Zhang (Eds.). Association for Computational Linguistics, Abu Dhabi, United Arab Emirates, 4553–4569. doi:10.18653/v1/2022.emnlp-main.302

[34] Ethan Perez, Patrick Lewis, Wen-tau Yih, Kyunghyun Cho, and Douwe Kiela. 2020. Unsupervised Question Decomposition for Question Answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Bonnie Webber, Trevor Cohn, Yulan He, and Yang Liu (Eds.). Association for Computational Linguistics, Online, 8864–8880. doi:10.18653/v1/2020.emnlp-main.713

[35] Mohammadreza Pourreza and Davood Rafiei. 2023. DIN-SQL: Decomposed In-Context Learning of Text-to-SQL with Self-Correction. In *Advances in Neural Information Processing Systems*, A. Oh, T. Naumann, A. Globerson, K. Saenko, M. Hardt, and S. Levine (Eds.), Vol. 36. Curran Associates, Inc., 36339–36348. https://proceedings.neurips.cc/paper_files/paper/2023/file/72223cc66f63ca1aa59edaec1b3670e6-Paper-Conference.pdf

[36] Mohammadreza Pourreza, Davood Rafiei, Yuxi Feng, Raymond Li, Zhenan Fan, and Weiwei Zhang. 2024. SQL-Encoder: Improving NL2SQL In-Context Learning Through a Context-Aware Encoder. arXiv:2403.16204 [cs.CL] https://arxiv.org/abs/2403.16204

[37] Bowen Qin, Binyuan Hui, Lihan Wang, Min Yang, Jinyang Li, Binhua Li, Ruiying Geng, Rongyu Cao, Jian Sun, Luo Si, Fei Huang, and Yongbin Li. 2022. A Survey on Text-to-SQL Parsing: Concepts, Methods, and Future Directions. arXiv:2208.13629 [cs.CL] https://arxiv.org/abs/2208.13629

[38] Abdul Quamar, Vasilis Efthymiou, Chuan Lei, and Fatma Özcan. 2022. Natural Language Interfaces to Data. *Foundations and Trends® in Databases* 11, 4 (2022), 319–414. doi:10.1561/1900000078

[39] Nils Reimers and Iryna Gurevych. 2019. Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics. https://arxiv.org/abs/1908.10084

[40] Qun Sui and Sujit K Ghosh. 2024. Entropy-Based Subsampling Methods for Big Data. *Journal of Statistical Theory and Practice* 18, 2 (2024), 24.

[41] Shayan Talaei, Mohammadreza Pourreza, Yu-Chen Chang, Azalia Mirhoseini, and Amin Saberi. 2024. CHESS: Contextual Harnessing for Efficient SQL Synthesis. arXiv:2405.16755 [cs.LG] https://arxiv.org/abs/2405.16755

[42] CodeGemma Team, Heri Zhao, Jeffrey Hui, Joshua Howland, Nam Nguyen, Siqi Zuo, Andrea Hu, Christopher A. Choquette-Choo, Jingyue Shen, Joe Kelley, Kshitij Bansal, Luke Vilnis, Mateo Wirth, Paul Michel, Peter Choy, Pratik Joshi, Ravin Kumar, Sarmad Hashmi, Shubham Agrawal, Zhitao Gong, Jane Fine, Tris Warkentin, Ale Jakse Hartman, Bin Ni, Kathy Korevec, Kelly Schaefer, and Scott Huffman. 2024. CodeGemma: Open Code Models Based on Gemma. arXiv:2406.11409 [cs.CL] https://arxiv.org/abs/2406.11409

[43] Enrico Toniato, Abdel Labbi, Katya Mirylenka, Christoph Miksovic Czasch, Thomas Gschwind, Paolo Scotton, Francesco Fusco, and Diego Antognini. 2023. FlowPilot: An LLM-powered system for enterprise data integration. In *Annual Conference on Neural Information Processing Systems*.

[44] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 6000–6010.

[45] Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, Linzheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2024. MAC-SQL: A Multi-Agent Collaborative Framework for Text-to-SQL. arXiv:2312.11242 [cs.CL] https://arxiv.org/abs/2312.11242

[46] Bailin Wang, Richard Shin, Xiaodong Liu, Oleksandr Polozov, and Matthew Richardson. 2020. RAT-SQL: Relation-Aware Schema Encoding and Linking for Text-to-SQL Parsers. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. 7567–7578.

[47] Tomer Wolfson, Daniel Deutch, and Jonathan Berant. 2022. Weakly Supervised Text-to-SQL Parsing through Question Decomposition. In *Findings of the Association for Computational Linguistics: NAACL 2022*, Marine Carpuat, Marie-Catherine de Marneffe, and Ivan Vladimir Meza Ruiz (Eds.). Association for Computational Linguistics, Seattle, United States, 2528–2542. doi:10.18653/v1/2022.findings-naacl.193

[48] Quan Xiao, Debarun Bhattacharjya, Balaji Ganesan, Radu Marinescu, Katsiaryna Mirylenka, Nhan Pham, Michael Glass, and Junkyu Lee. 2025. The Consistency Hypothesis in Uncertainty Quantification for Large Language Models. In *The Conference on Uncertainty in Artificial Intelligence (UAI) 2025*.

[49] Yuanzhen Xie, Xinzhou Jin, Tao Xie, MingXiong Lin, Liang Chen, Chenyun Yu, Lei Cheng, ChengXiang Zhuo, Bo Hu, and Zang Li. 2024. Decomposition for Enhancing Attention: Improving LLM-based Text-to-SQL through Workflow Paradigm. *arXiv preprint arXiv:2402.10671* (2024).

[50] Sun Yang, Qiong Su, Zhishuai Li, Ziyue Li, Hangyu Mao, Chenxi Liu, and Rui Zhao. 2024. SQL-to-Schema Enhances Schema Linking in Text-to-SQL. arXiv:2405.09593 [cs.DB] https://arxiv.org/abs/2405.09593

[51] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir Radev. 2018. Spider: A Large-Scale Human-Labeled Dataset for Complex and Cross-Domain Semantic Parsing and Text-to-SQL Task. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, Ellen Riloff, David Chiang, Julia Hockenmaier, and Jun'ichi Tsujii (Eds.). Association for Computational Linguistics, Brussels, Belgium, 3911–3921. doi:10.18653/v1/D18-1425

[52] Bin Zhang, Yuxiao Ye, Guoqing Du, Xiaoru Hu, Zhishuai Li, Sun Yang, Chi Harold Liu, Rui Zhao, Ziyue Li, and Hangyu Mao. 2024. Benchmarking the Text-to-SQL Capability of Large Language Models: A Comprehensive Evaluation. arXiv:2403.02951 [cs.CL] https://arxiv.org/abs/2403.02951

[53] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating Structured Queries from Natural Language using Reinforcement Learning. arXiv:1709.00103 [cs.CL] https://arxiv.org/abs/1709.00103

[54] Denny Zhou, Nathanael Scharli, Le Hou, Jason Wei, Nathan Scales, Xuezhi Wang, Dale Schuurmans, Olivier Bousquet, Quoc Le, and Ed Huai hsin Chi. 2022. Least-to-Most Prompting Enables Complex Reasoning in Large Language Models. *ArXiv* abs/2205.10625 (2022). https://api.semanticscholar.org/CorpusID:248986239

[55] Xiaohu Zhu, Qian Li, Lizhen Cui, and Yongkang Liu. 2024. Large Language Model Enhanced Text-to-SQL Generation: A Survey. arXiv:2410.06011 [cs.DB] https://arxiv.org/abs/2410.06011