⋆ STARQA: A Question Answering Dataset for Complex Analytical Reasoning over Structured Databases

Mounica Maddela[1], **Lingjue Xie**[1], **Daniel Preoţiuc-Pietro**[1], **Mausam**[2*]
[1]Bloomberg
[2]Yardi School of Artificial Intelligence, Indian Institute of Technology, Delhi
{mmaddela3,lxie91,dperotiucpie}@bloomberg.net, mausam@cse.iitd.ac.in

Abstract

Semantic parsing methods for converting text to SQL queries enable question answering over structured data and can greatly benefit analysts who routinely perform complex analytics on vast data stored in specialized relational databases. Although several benchmarks measure the abilities of text to SQL, the complexity of their questions is inherently limited by the level of expressiveness in query languages and none focus explicitly on questions involving complex analytical reasoning which require operations such as calculations over aggregate analytics, time series analysis or scenario understanding. In this paper, we introduce STARQA, the first public human-created dataset of complex analytical reasoning questions and answers on three specialized-domain databases. In addition to generating SQL directly using LLMs, we evaluate a novel approach (TEXT2SQLCODE) that decomposes the task into a combination of SQL and Python: SQL is responsible for data fetching, and Python more naturally performs reasoning. Our results demonstrate that identifying and combining the abilities of SQL and Python is beneficial compared to using SQL alone, yet the dataset still remains quite challenging for the existing state-of-the-art LLMs.

1 Introduction

Text to SQL semantic parsing approaches enable intuitive and efficient human interaction with structured databases through natural language queries. They alleviate the need for the user to fully comprehend the underlying DB schema or the query language used to answer their question. Large language models (LLMs) exhibit very good performance in this task in constrained settings, measured by over 90% execution match accuracies on popular data sets such as Spider (Yu et al., 2018) and WikiSQL (Zhong et al., 2017).

An important user base for these methods can be analysts whose goals are to perform analyses involving complex analytics that require reasoning over large, proprietary datasets, in order to obtain quantitative insights that can drive data-driven decision making. Examples include economic analysts deciding on a country's policies, financial analysts making investment decisions, operations researchers that improve business operations or data-driven sports analysts making match or acquisition strategies. It is estimated that about 52% of professional developers regularly use SQL, and over one-third must run very complex queries efficiently (StackOverflow, 2023; PremAI, 2024). However, almost all of the 36 benchmark datasets, surveyed in Liu et al. (2024), contain less than two `select` clauses (suggesting lack of nested queries and complex set operations) and less than one mathematical, scalar or aggregation function per query.

In response, we introduce a novel dataset STARQA – **St**ructured Data **A**nalytics & **R**easoning **Q**uestion **A**nswering. It contains 362 realistic complex analytics and reasoning questions and answers over publicly available databases from three domains – movies (`IMDb`), sports (`ES`), and E-commerce (`OL`). Each question requires one or more types of reasoning, such as math operations, time-series analysis, nested queries, calculations over aggregate analytics, temporal reasoning, and scenario understanding. Table 1 illustrates examples of questions in STARQA and their categories.

The types of reasoning in STARQA raise a question about whether SQL syntax alone is the best choice for obtaining the correct answer. Cases that require operations such nested loops, complex conditional logic or data manipulation can lead to very complex SQL syntax and long queries, which can be cumbersome for a model to generate.

We propose TEXT2SQLCODE as a potential solution to decompose the task in multiple steps that

*Most of the work was done when the author was on a sabbatical at Bloomberg.

Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing, pages 34487–34499
November 4-9, 2025

interweave SQL for efficient data operations over databases and Python code for operations that are too burdensome to achieve using SQL, such as performing analytics, data wrangling and computations. A challenge of this solution approach is its multi-step nature that can lead to compounding errors. Thus, we explore its hybrid variants, where we apply TEXT2SQLCODE only on the subset of difficult reasoning questions as automatically identified through the Text2SQL parser's outputs.

We extensively benchmark recent open-weight and commercial LLMs, including reasoning models, on STARQA. Results show that STARQA is challenging for all methods and models tested, with the median performance (across six LLMs) of a standalone Text2SQL model reaching only 36.5% execution accuracy. However, our best performing hybrid approaches (HYBRID$_{single}$ and HYBRID$_{multi}$) achieve improved performance on this task and significantly outperform Text2SQL, with median performance of 45.4% and 44.5% respectively. We also find that most of our questions cannot be solved by existing LLMs with access to web search tools – reinforcing the unique potential of large-scale, rich and potentially private proprietary structured data. Finally, our analyses highlight the nature of the errors and suggest directions for future research in building stronger reasoning systems over structured databases. We release STARQA publicly and freely for research.[1]

2 Related Work

Semantic parsing techniques convert natural language inputs into formal representations, such as λ-calculus for robotics (Williams et al., 2018), SQL for structured databases (Katsogiannis-Meimarakis and Koutrika, 2023), SPARQL for KBs (Patidar et al., 2024), Cypher for graph data (Ozsoy et al., 2025), and other low-resource languages for specific applications (Dutta et al., 2024). While our dataset is specific to SQL, our methods and observations could be more generally applicable.

The latest Text2SQL systems obtain nearly 90% performance on popular benchmarks that study constrained settings of the task such as WikiSQL[2] and Spider.[3] Various benchmarks highlight challenges in specific domains such as finance (Sen et al., 2020), accounting (Kumar et al., 2024) and science (Zhang et al., 2023). Other works study generic notions of complexity such as variety in SQL dialects and interaction with real-world enterprise databases (Lei et al., 2024), complex and large database schemas (Li and Jagadish, 2014; Sen et al., 2020), issues with entity linking (Wang et al., 2022), and use of external knowledge (Li et al., 2021). After surveying 36 benchmark datasets for Text2SQL, Liu et al. (2024) identify reasoning as a key underexplored aspect of the task. STARQA is developed to bridge this gap.

Most related, `Archer` (Zheng et al., 2024) contains questions on arithmetic, commonsense, and counterfactual reasoning; however, they can all be solved by the basic SQL language and do not require any procedural extensions or scripting languages. This limits the overall complexity of the questions – they do not require string manipulations, complex cell processing, or statistical functions – some of the features of STARQA. SPIDER 2.0 (Lei et al., 2024), on the other hand, combines many levels of complexity in its questions on real-world enterprise databases and is best suited for developing a complete agentic workflow with multiple components. However, it is not an ideal dataset for evaluating analytical reasoning abilities and is not annotated with nature of reasoning required.

The main LLM-based approach for Text2SQL systems (Shi et al., 2024) includes building schema and entity linkers (Wang et al., 2025b) whose outputs, along with retrieved exemplars (Gurawa and Dharmik, 2025), are input to an LLM for SQL generation. CoT prompting (Tai et al., 2023), self-consistency over multiple generations (Sun et al., 2023), self-correction (Pourreza and Rafiei, 2023), and iterative repair (Sawhney et al., 2025) are also used to improve generation quality.

Our proposed solution is influenced by the rapid progress in code generation systems (Zan et al., 2023), and investigates whether SQL and Python can be effectively combined. To the best of our knowledge, there is limited exploration of this synergy in literature. TPTU (Ruan et al., 2023) studies this but with somewhat artificial prompts where, often a *simple* mathematical operation (e.g., log or factorial) is applied over a statistic outputted by an SQL query. Another work (Sui et al., 2023) assesses Text2Python as an alternative to Text2SQL, but finds that former performs worse, primarily because of high generation length requirements. It does not explore their combination.

Our approach is also related to table-augmented generation (Biswal et al., 2024) where SQL results

[1] https://zenodo.org/records/17157169
[2] https://github.com/salesforce/WikiSQL
[3] https://yale-lily.github.io/spider

are provided in context, for LLMs to generate a final answer. This succeeds only when SQL outputs are short, and processing post-SQL is limited. It does not study combinations with other languages.

3 The STARQA Dataset

We create a new dataset to study complex analytical reasoning question answering over SQL databases. The dataset is built to contain questions that require one or more of the categories described in Table 1. The STARQA dataset consists of the following: (1) questions, (2) their answers, (3) categories required to answer a question, and (4) reference code used to obtain the gold answer. The code is only presented as a reference to verify the answer and is not used in any experiments in this paper.

3.1 Databases

To create STARQA, we use three complex and large databases from the movies, sports, and E-commerce domains. We use only three DBs so that data creation is invested primarily on creating a diverse set of questions with depth in reasoning and analytics, which is the goal of this dataset. This can be achieved only if the annotators understand the data, the domain and DB organization deeply enough to ask and answer interesting questions, limiting the number of databases we can scale to. As constructed, we expect STARQA to be useful primarily for development and evaluation, not for training.

IMDb. This database[4] is provided by IMDb and contains historical information about over 10 million movie titles, including TV series, their crews, biographical information about crew members, movie ratings on IMDb and number of votes. In total, the IMDb database has 7 tables, with a table having up to 9 columns and a total of 11 million rows.

ES. This database[5,6] consists of soccer (football) statistics from the top-flight leagues of 11 European countries between 2008 and 2016. In total, the database includes information on over 25,000 matches, 10,000 players, their attributes, match events, line-ups, formations and betting odds. In total, the database has 16 tables, with tables having up to 115 columns and 26,000 rows.

OL. This database[7] consists of 100k orders from the Olist - online store in Brazil, from 2016 to 2018. It provides a multi-dimensional view of orders, including information on order status, price, payment, freight performance, customer location, product attributes, and customer reviews. In total, the database has 9 tables, with tables having up to 9 columns and 112,000 rows.

3.2 Dataset Creation

STARQA is manually created in its entirety by the authors of the paper, who all have training and experience in programming and NLP, intermediate or higher knowledge of SQL, advanced knowledge of Python, obtained advanced knowledge of the databases and their structure, and have a personal interest in the domains represented in the dataset.

Our aim is to build a dataset of questions that are both realistic, akin to those that an analyst working with these databases may ask, and that cover as many of the complex analytical reasoning categories described in Table 1 as possible.

We first start by selecting, for each database, the categories of questions that would be natural to the domain and the underlying database structure. After all questions were written, these were reviewed by the other authors and edited until they satisfied the following criteria: correctness (the question-answer pair is syntactically and semantically correct), credibility (if the question is one an analyst might ask), specificity (lack of ambiguity in the information that is requested) and clarity of the expected output and its format.

We construct the STARQA dataset to facilitate exact automatic evaluation of the performance of systems. Each question clearly states the expected returned fields, and the precision of the numbers in case a fraction or percentage is expected. The answer to each question is formatted in a canonical format represented by a list of tuples, with each item in the list representing a correct answer to the question. Each question can have no answer, one unique answer tuple or more. Each tuple may have one or more constituents – no specific order for them is imposed within a tuple, and set-match is used for evaluation. We also aim to limit ambiguities in question formulation wherever possible, such that no model is penalized for picking different interpretations e.g., we replace *Find the actor* with *Find the actor, male or female*. While these

[4] https://developer.imdb.com/non-commercial-datasets/
[5] https://www.kaggle.com/datasets/hugomathien/soccer
[6] https://www.kaggle.com/datasets/jiezi2004/soccer
[7] https://www.kaggle.com/datasets/olistbr/brazilian-ecommerce

Category	Example	#Samples
Stat./Math. Operations	*Is there a **statistically significant correlation (use Spearman's)** between a movie's runtime and its average rating?*	85
Analytics over Non-entries in Database	*What is the **most common uncased prefix (first 3 characters)** across all movie titles released in the 1990s, removing any preceding articles (a, an, the)?*	20
Nested Queries	*Find the actor/actress with the **most appearances in movies that are in the top 10% by rating every decade**.*	147
String Manipulation	*Which movie title has the highest number of **distinct characters (ignoring case and spaces)**?*	46
Calculations over Aggregate Analytics	*Percentage (no decimals) for **a team to win the league if they are 3 points or more ahead of all the other teams at the end of January***	100
Complex Columns	*Identify the most popular **co-occurring genre pairs** among TV series*	16
Temporal Reasoning	*Print the number of directors that got an 9.0 rating for their debut in **1980s**.*	32
Complex Filtering	*Which director has directed movies in **at least ten different languages and has an average movie rating above 8.5**?*	36
Unit Conversions	*How many years would it take to watch every movie in the database?*	9
Scenario Understanding	*Which teams that won the league would not have won it **if a win would have been only 2 points**.*	16
Time Series Analysis	*Teams in the English Premier League that **lost four matches in a row after winning four in a row**.*	55
Commonsense Knowledge	*The longest number of months either team name was undefeated in the **Old Firm** between 2008-2016?*	18

Table 1: Reasoning categories in STARQA, along with an example and number of questions that exhibit this category. A question may have multiple categories. Output formatting instructions in the questions are omitted for brevity.

requirements may make the question appear less natural, they maximize the ability to accurately benchmark the abilities of a model in answering the questions correctly.

The goal of STARQA is to measure analytical reasoning abilities. A common challenge in other Text2SQL datasets that include reasoning questions (Lei et al., 2024; Zheng et al., 2024) is entity linking in a new database and adapting the model to the new database schema. We consider these as orthogonal challenges to our goal. In order for them to have a limited impact in our dataset, we normalize the mentions of specific entities (e.g., actors, league or team names) to a common and unambiguous name.

The answers to all the questions were obtained through executing a combination of SQL and Python code written by the dataset creators that is released with the dataset. The authors spent, after data collection and data preparation, on average between 15 to 30 minutes writing the answer to a single question.

3.3 Dataset Statistics

Overall, the STARQA dataset consists of 362 data points, with 100 questions on `IMDb`, 162 questions on `ES`, and 100 questions from `Olist`. Out of these, 55 questions have a single numerical answer, 8 questions have a null answer, and most questions (299) have tuples as an answer. The average number of tuples in an answer is 2.35 (up to 66) and the average number of constituents in a tuple is 2.43 (up to 8). The distribution over the types of analytical reasoning categories and example questions are presented in Table 1.

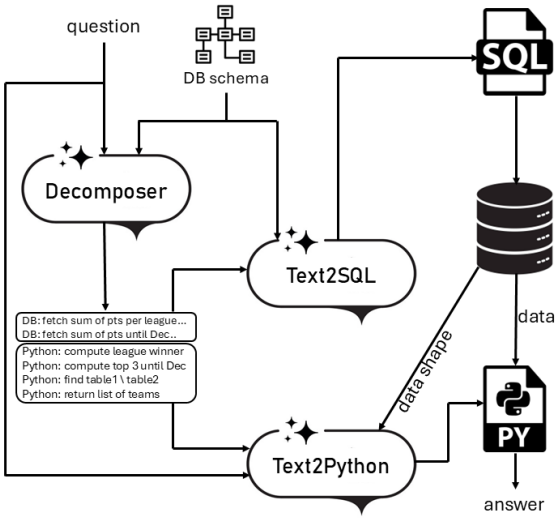

Figure 1: The architecture of TEXT2SQLCODE$_{multi}$

4 The TEXT2SQLCODE Approach

We propose TEXT2SQLCODE, which combines the use of text to SQL semantic parser (Text2SQL) and a Python code generator (Text2Python) for answering the analytical reasoning questions. The underlying insight is that such questions can typically be split into data fetching and processing or computation steps. SQL is required for data fetching, as the data is provided in relational databases. If the required analytical reasoning is complex, a procedural language (such as Python) may be needed due to the limited expressiveness of the SQL language in performing some operations. Consequently, a split may be desirable, where data fetching, and perhaps simple processing, is conducted in SQL and complex processing (if needed) is done in Python.

We operationalize this by designing a three-step workflow (Figure 1): (1) a decomposer, (2) a Text2SQL model, and (3) a Text2Python model. We implemented two variants of the workflow: TEXT2SQLCODE$_{multi}$ and

TEXT2SQLCODE$_{single}$. TEXT2SQLCODE$_{multi}$ uses separate LLM calls for each step with specific prompts and output formats. The LLM outputs for each step are post-processed for the next step. In contrast, TEXT2SQLCODE$_{single}$ uses a single prompt to perform decomposition and generates a complete Python function wrapped around the SQL query. We note that the idea of question decomposition has been explored within Text2SQL (Pourreza and Rafiei, 2023; Wang et al., 2025a), though not in combination with Python.

Decomposer: The goal here is to decompose the user question into a series of steps, along with annotation on whether they are Text2SQL steps or Text2Python steps. We provide the DB schema – tables, column types and descriptions, primary and foreign keys, sample rows per table – in the prompt. We also provide a handful of positive and negative exemplars (from another domain, for a fair comparison with other approaches).

We provide these additional guidelines in the prompt: (1) Multiple Text2SQL steps should not be dependent on each other, and should be independently executable; (2) Python may be omitted for simpler questions; (3) Each step should be in natural language and should be prefixed with either "`Text2SQL: `" or "`Python: `". We then ask the LLM to perform chain of thought and provide its final output. In Figure 1, the two types of steps (SQL and Python) in the decomposition are shown in two separate boxes.

Text2SQL: We generate an SQL query for each Text2SQL step identified by the decomposer. For the TEXT2SQLCODE$_{multi}$ approach, a separate Text2SQL call is made for each identified step. We provide the database schema and the specific text of the step. The generated queries are then executed to fetch data into a dataframe. If the query fails, we reprompt the LLM for corrections, up to three times. For the TEXT2SQLCODE$_{single}$, the LLM directly generates SQL queries that are embedded within the final Python function, creating a single-pass solution.

Text2Python: This step handles the complex analytical reasoning that goes beyond reasoning abilities. In the TEXT2SQLCODE$_{multi}$ approach, this step is executed after all the necessary data has been fetched from the Text2SQL calls. We construct a prompt with the original user question, the decomposition, and the shape of each dataframe generated via the SQL calls from the previous step. The shape includes the names of columns and a few sample rows of data (similar to Maamari and Mhedhbi (2024)). We prompt the model to generate a single Python function with a specific signature:

```
compute_result(listOfDFs:
List[DataFrame]) -> List[Tuple].
```

We then run the generated Python code on the full dataframes retrieved in the Text2SQL step(s). TEXT2SQLCODE$_{single}$ also implements a similar Python function with the DB path as the input:

```
compute_result(db_path) -> List[Tuple].
```

This function also contains the embedded SQL queries. For both cases, if the code runs successfully, we output the result, else, we reprompt the LLM for correction (maximum of three times).

We note this framework can be extended by using retrieval when the DB is too large for the schema to fit in the prompt or by using an explicit entity linking step. We eschew these in our experiments for simplicity, as they are not required for STARQA.

4.1 A Text2SQL-Text2SQLCode Hybrid

Our early experiments indicated that LLMs cannot assess well for which questions to use the decomposition and invoke Python. We thus propose a hybrid approach where we only run TEXT2SQLCODE when Text2SQL can not reliably provide a result. We approximate this by using self-consistency (Arora et al., 2023) over three runs as a proxy for question difficulty: if two of the three runs provide the same answer, that answer is taken as the prediction; otherwise, the TEXT2SQLCODE is invoked and its answer is taken as the prediction. This system should be able to gain meaningfully from the strengths of both Text2SQL and TEXT2SQLCODE on more complex questions. We implement two hybrid variants HYBRID$_{multi}$ and HYBRID$_{single}$ corresponding to TEXT2SQLCODE$_{multi}$ and TEXT2SQLCODE$_{single}$ respectively.

5 Experimental Setup

5.1 System Details

As baselines, we use the standalone Text2SQL component as described in Section 4, using the same experimental and task setup. Additionally, we also implement a self-consistency (Wang et al., 2023) approach (KText2SQL + SC) where we output the majority of K answers, or one at random if there is no majority. For our experiments, we use $K = \{3, 5\}$. We test all approaches on a variety of current state-of-the-art LLMs including both

open- and closed-sourced models, including reasoning models. We use the following closed-source (API) models: Claude 3.7, GPT 4.1, GPT o1, GPT o3-mini with their latest versions as of 14 September 2025, and the following open-weight models: DeepSeekV3.1 and Qwen2.5. Out of these, GPT o1 and GPT o3-mini are reasoning models. We use the default temperature for all the models.

Prompts: We construct one prompt for each database. The prompts are formatted to contain the following information:

- A description of the task;
- The database schema, including table names, column names, data types and column definitions;
- Data samples for each table, such that the LLM is familiar with data types and formats;
- A list of all possible values for all fields containing categorical data and their description;
- We instruct that the output should be a list of tuples without any additional names or descriptions, and should not output any additional information, even if relevant. These instructions are added to ensure that the output can be auto-evaluated, as LLMs have a tendency to output additional information.

The above metadata is provided in order for the model to obtain as much grounding in the data schema and values as possible. All prompt templates from our experiments are in the supplementary material.

5.2 Evaluation Metrics

We test all our approaches on accuracy of execution by running the generated code and comparing the outputs with the reference answer. The output format (list of tuples) is consistent across all questions and is provided as instructions in the prompt. A manual inspection of the output of the model shows this format is widely respected by all models. The execution match metric is insensitive to the order in which the tuples are provided, or the order in which a tuple is constructed. Given these considerations, we are confident that the execution accuracy is an accurate representation of system performance.

5.3 Additional Comparisons

Knowledge: The LLMs' parametric knowledge may contain information about the questions and answers present in STARQA, especially if these questions are about information available online. As an additional comparison point to understand the complexity of the STARQA, we test the ability of the models to answer these questions directly from their parametric knowledge, by providing explicit instructions in the prompt to the model to not use SQL or code.

LLMs using Search: We aim to show the performance of state-of-the-art LLM-based agents that can use reasoning, search and other tools in answering questions. These systems can access information and statistics available on websites, or identify articles related to similar questions available online, in order to provide answers to the STARQA questions. Albeit, this is not a comparable method to other approaches studied in the paper, as it has access to additional information, we test this in order to understand the value differential that exists in structured data sources. As an exponent of such system, we use Gemini Pro 2.5 (Gemini-2.5-Pro-Preview-05-06) on the week of May 12th, as this tops the Chatbot Arena Leaderboard.[8] Given the model does not always adhere to the output format, we additionally process the outputs manually for all questions to obtain its performance score.

`Archer` Dataset: To demonstrate the generalizability of our TEXT2SQLCODE approaches, we also include results on the English validation set of the `Archer` dataset (AR) (Zheng et al., 2024), an existing text to SQL dataset. This dataset contains 104 questions from 2 databases.

6 Results

Table 2 reports the performance of all six LLMs tested on the STARQA and `Archer` datasets using execution accuracy, each score is an average of three runs.

Overall performance on STARQA is moderate. The best result obtained on the overall dataset is 48.3%. This highlights that the STARQA represents a challenging dataset for current state-of-the-art LLMs, especially because there are no challenges regarding entity linking and adapting to a new DB schema present like in other datasets (Lei et al., 2024; Zheng et al., 2024). The best performing model overall is GPT o3-mini, but the gaps between models (Claude 3.7, GPT 4.1, GPT o1, GPT o3-mini) are narrow, within 4% overall. Results on the three sections of the dataset show other models performing best (GPT 4.1 for `IMDb`, GPT-o3-mini for `ES`, and GPT-o1 for `OL`), with no clear pattern of what the superior model is. Interestingly, despite reasoning models claiming better performance on

[8] https://lmarena.ai/

Model	Method	STARQA	IMDB	ES	OL	Calls	AR
Claude 3.7	Knowledge	4.9	1.3	8.8	2.3	**1**	2.6
	Text2SQL	35.7	29.3	43.4	29.7	**1**	29.5
	Text2SQLCode$_{single}$	39.9	**50.7**	34.8	37.3	**1**	32.0
	Text2SQLCode$_{multi}$	44.0	45.3	48.8	35.0	2.8	**36.2**
	3Text2SQL + SC	37.8	30.3	46.1	32.0	3	30.1
	5Text2SQL + SC	38.9	36.0	45.2	31.7	5	29.2
	Hybrid$_{single}$	42.5	41.7	46.9	**36.5**	3.4	32.0
	Hybrid$_{multi}$	**44.2**	42.3	**50.8**	35.3	4.1	32.4
	Oracle$_{single}$	51.5	57.3	52.5	44.0	-	48.4
	Oracle$_{multi}$	52.7	54.7	56.4	44.7	-	45.8
GPT 4.1	Knowledge	6.6	0.0	13.4	2.3	**1**	6.1
	Text2SQL	37.2	43.0	40.5	26.0	**1**	28.2
	Text2SQLCode$_{single}$	40.4	57.7	32.3	**36.3**	**1**	30.4
	Text2SQLCode$_{multi}$	40.9	53.0	36.8	35.3	2.9	**41.3**
	3Text2SQL + SC	40.5	47.7	43.2	29.0	3	28.9
	5Text2SQL + SC	39.7	44.4	43.7	28.4	5	29.5
	Hybrid$_{single}$	**46.4**	**60.0**	44.7	35.7	3.2	32.7
	Hybrid$_{multi}$	44.6	55.0	**45.5**	32.7	4.1	40.7
	Oracle$_{single}$	53.8	65.7	51.2	46.0	-	38.5
	Oracle$_{multi}$	53.1	62.7	52.1	45.3	-	46.2
GPT o1	Knowledge	11.6	1.7	24.5	0.7	**1**	5.5
	Text2SQL	39.1	41.3	43.2	30.3	**1**	24.7
	Text2SQLCode$_{single}$	48.0	**59.0**	47.9	38.0	**1**	34.9
	Text2SQLCode$_{multi}$	43.8	50.3	46.3	33.3	2.6	32.7
	3Text2SQL + SC	41.8	44.3	46.1	32.2	3	25.0
	5Text2SQL + SC	41.5	43.4	46.4	31.7	5	25.3
	Hybrid$_{single}$	**48.2**	50.0	**52.7**	**38.5**	3.3	**36.2**
	Hybrid$_{multi}$	45.7	49.3	49.2	36.3	3.8	34.6
	Oracle$_{single}$	56.8	66.1	57.8	45.7	-	38.1
	Oracle$_{multi}$	52.8	60.0	54.5	43.0	-	36.9
GPT o3-mini	Knowledge	3.5	3.0	5.1	1.5	**1**	5.8
	Text2SQL	43.8	47.3	50.0	30.3	**1**	26.9
	Text2SQLCode$_{single}$	45.9	**59.9**	44.0	**35.0**	**1**	34.0
	Text2SQLCode$_{multi}$	44.1	51.0	46.7	33.0	2.6	**36.5**
	3Text2SQL + SC	45.2	48.3	52.1	31.0	3	27.9
	5Text2SQL + SC	46.0	50.8	51.9	31.7	5	29.2
	Hybrid$_{single}$	**48.3**	54.0	53.5	34.3	3.2	32.7
	Hybrid$_{multi}$	47.5	53.0	**53.7**	32.0	3.7	35.6
	Oracle$_{single}$	55.9	66.0	57.4	43.3	-	37.8
	Oracle$_{multi}$	52.8	58.3	56.8	40.7	-	40.1
DeepSeek V3.1	Knowledge	3.1	1.3	4.5	2.7	**1**	1.9
	Text2SQL	29.9	30.0	34.2	22.7	**1**	26.9
	Text2SQLCode$_{single}$	28.7	49.3	18.1	25.3	**1**	25.3
	Text2SQLCode$_{multi}$	43.6	52.5	44.4	33.3	2.81	**36.2**
	3TextSQL + SC	32.2	33.0	37.2	23.3	3	28.9
	5Text2SQL + SC	32.6	32.4	36.9	25.7	5	28.2
	Hybrid$_{single}$	37.3	49.0	36.2	27.3	3.5	26.3
	Hybrid$_{multi}$	**44.4**	**53.6**	**45.1**	**34.1**	4.41	34.6
	Oracle$_{single}$	42.5	59.0	38.9	31.7	-	36.2
	Oracle$_{multi}$	51.7	61.5	51.9	41.7	-	41.3
Qwen 2.5	Knowledge	0.71	0.3	0.6	1.3	**1**	0
	Text2SQL	11.4	13.3	11.7	9.0	**1**	12.8
	Text2SQLCode$_{single}$	12.1	13.3	9.7	14.7	**1**	13.1
	Text2SQLCode$_{multi}$	15.7	22.0	11.9	15.7	3	13.3
	3Text2SQL + SC	12.2	14.7	11.9	10.3	3	14.1
	5Text2SQL + SC	12.2	14.2	12.2	10.2	5	13.8
	Hybrid$_{single}$	16.7	19.3	13.2	19.7	3.7	14.1
	Hybrid$_{multi}$	**18.3**	**24.0**	**13.6**	**20.3**	5.2	**14.4**
	Oracle$_{single}$	20.6	23.0	19.1	20.7	-	23.7
	Oracle$_{multi}$	24.0	30.7	21.2	22.0	-	18.6
Gemini Pro 2.5	Search	31.2	4.0	65.4	3.0	-	-

Table 2: Execution accuracy (average over three runs) and the average #LLM calls for 6 LLMs with zero-shot prompting on the STARQA dataset, its three underlying databases, and the Archer (AR) dataset. Here, Oracle refers to combining the best output of TextSQL and Text2SQLCode post-hoc. Underline represents the best overall result for that dataset, and bold shows the best result for model-dataset pair. The Calls column refers to the average number of LLM calls used by each method.

coding tasks such as this, the two reasoning models tested (GPT o1 and GPT o3-mini) do not perform substantially better overall on STARQA than non-reasoning models.

Overall, a clear pattern is that the closed-source models outperform open-source models we tested. This is especially evident when using Text2SQL alone. In this setup, the GPT o3-mini model performs substantially better (+4.7% overall) than any other model.

TEXT2SQLCODE performs better than Text2SQL alone. For all the models, the performance of the TEXT2SQLCODE methods (both single and multi-step versions) are higher than the Text2SQL method alone, with the overall accuracy improvements ranging up to 14.5% on STARQA for DeepSeek and 13.1% on Archer for GPT-4.1. Overall, these results highlight that decomposing the task into SQL and Python code execution is a promising direction to improve the performance of the model. We see consistent improvements not only on our own dataset but also on Archer, which demonstrates the generalizability of our approach. We highlight that each TEXT2SQLCODE$_{multi}$ run requires multiple LLM calls, between 2 and 3, depending on whether Python code is generated.

Self-consistency for Text2SQL improves performance. Self-consistency over three Text2SQL executions improves performance of the system, in line with past work. Performance improvements are on average 2.1% and range between 0.8% and 3.3% percent. However, we do not always see improvements between three and five executions. In fact, self-consistency with five executions shows performance degradation on STARQA for GPT-o1 and GPT-4.1. This shows that simply increasing the number of executions is not enough to improve the overall performance. Note that the improvements come at a cost trade-off, as three or five LLM calls are necessary. Performance of self-consistency, even with Text2SQL alone, is, in many cases, comparable to using TEXT2SQLCODE and comes at a similar number of LLM calls.

Different TEXT2SQLCODE approaches perform well on different domains. Hybrid methods are best for overall accuracy on the STARQA and ES, but performance on other domains and Archer is mixed. For 3 of 6 models, TEXT2SQLCODE$_{single}$ is the most accurate on IMDb, while TEXT2SQLCODE$_{multi}$ is most accurate on Archer for 4 out of 6 models. These experimental results show that the TEXT2SQLCODE decomposition should be deployed selectively on questions. An additional indicator is the Oracle method performance, which takes post-hoc the best prediction from the Text2SQL and the TEXT2SQLCODE methods. This achieves better performance than any of the two systems by substantial margins from 3%-9%. These suggest that, while the model can produce the correct answer, it

is hindered by its ability to estimate confidence in its predictions. We study this further in the next section.

Methods not using structured data do not perform well on STARQA. We conduct two experiments where the structured database is not used, in order to measure the value this data brings to the task. As expected, using the model's internal knowledge to answer the questions leads to a low overall performance below 10%, with the exception of GPT o1 at 11.6%. Even if the questions are what analysts would ask and thus could be present in the original data used to train the model, usually simple additional constraints (e.g., time range filtering) is enough to incapacitate the model to produce the right answer.

Next, we check the results of the Gemini Pro 2.5 system. The system heavily relies on live searches over the web (usually more than 10 for our questions) to retrieve relevant articles and perform reasoning using the retrieved data in order to find the correct answer. Given that the questions in the `EuroSoccer` dataset are realistic records that people interested in this sport may ask themselves, as well as summary statistics (e.g., league tables) being readily available online, the performance of the system on the `EuroSoccer` section is good and better than all models that use structured data (65.4%). Still, there are still substantial gaps in performance, especially on specific categories such as time series analysis. On the other hand, on the `IMDb` and `Olist` sections of the dataset, the performance is very low. This is caused by the nature and difficulty of the questions, which do not ask for well-known records, and because the size of the search space is much vaster than for `EuroSoccer`, which is only dealing with 11 leagues for 8 seasons.

Both these experiments highlight both the value that is stored in the structured data and that the dataset itself is a challenging dataset in general. Due to the type of the questions and the fine-grained data they require, we expect that this dataset could also represent a valuable general benchmark of LLMs both when using or not using the structured data for grounding.

6.1 Analysis on Use of Python Code

We further investigate the question routing performance of TEXT2SQLCODE and HYBRID, as results indicated that HYBRID performs substantially better likely because of more judicious use of Python. Recall that TEXT2SQLCODE has the ability to not use Python if it does not deem it necessary. We measure if this is indeed used for the more difficult questions or it is overused for easier questions, leading to potential loss of performance due to issues in decomposition. To understand this, Table 3 (column – 'Full') reports the percentage of questions that use Python in TEXT2SQLCODE$_{multi}$. Interestingly, we find that all non-reasoning models use Python for 80% or more questions, hinting at its overuse, given the Text2SQL performance is mostly in the 30-40% range.

Model	Method	Pct Questions w/ Python execution		
		Full	T2SQL correct	T2SQL incorrect
Claude 3.7	TEXT2SQLCODE$_{multi}$	85.2	-4.8	+2.5
	HYBRID$_{multi}$	34.2	-18.8	+9.7
GPT 4.1	TEXT2SQLCODE$_{multi}$	87.6	-4.7	+2.5
	HYBRID$_{multi}$	38.0	-17.7	+9.4
GPT o1	TEXT2SQLCODE$_{multi}$	53.2	-8.4	+4.1
	HYBRID$_{multi}$	23.0	-13.2	+6.5
GPT o3-mini	TEXT2SQLCODE$_{multi}$	54.9	-9.4	+5.7
	HYBRID$_{multi}$	19.6	-12.9	+7.8
DeepSeek V3.1	TEXT2SQLCODE$_{multi}$	79.3	-3.6	+1.6
	HYBRID$_{multi}$	41.2	-23.1	+10.1
Qwen 2.5	TEXT2SQLCODE$_{multi}$	97.8	-3.0	+0.3
	HYBRID$_{multi}$	68.5	-39.2	+4.6

Table 3: Post-hoc analysis showing the percentage of questions routed to the Python executor for three sets: full data, the subset where Text2SQL answer is correct and incorrect, respectively. For latter two, we report the relative value w.r.t. the full set value.

We then compute the same value, but for the two subsets of questions that Text2SQL gets correct or not. While the model can not know this at test time, an ideal model would be aware of its Text2SQL capabilities, and only invoke Python for questions that Text2SQL can not get right. We find that in TEXT2SQLCODE there is only $< 9\%$ relative change in either direction for the two subsets, meaning that TEXT2SQLCODE's routing of questions to Python is almost oblivious of the actual accuracy of Text2SQL on the question. This result holds across all six LLMs tested. These results substantiate why in several cases TEXT2SQLCODE underperforms Text2SQL (see Table 2), as TEXT2SQLCODE is a pipeline and errors can compound more than for Text2SQL, which needs to output a single SQL statement. This is perhaps expected, given that LLMs are generally not well calibrated (Arora et al., 2023; Stengel-Eskin and Van Durme, 2023; Liu et al., 2025).

The HYBRID system, on the other hand, uses Text2SQL system's own consistency as its proxy for difficulty. The same experiment performed on HYBRID shows that there is a up to 39% relative deviation in routing percentage on the two subsets,

leading the system to gain from the complementary strengths of the two systems.

6.2 Error Analysis

Text2SQL vs TEXT2SQLCODE: Our approaches excel in questions requiring: (1) Advanced string manipulation and regular expressions, (2) Conditional aggregation using derived categories, (3) A clear split between database access (SQL) and logic-heavy processing (Python), and (4) Large-scale joins and filtering in SQL, combined with dynamic grouping, per-row aggregation, and in-memory calculations in Python. We present two examples below.

Example 1: *For each genre with at least 100,000 known movies, compute the percentage of movies rated above 8.0 and rated below 5.0.*

This highlights the limits of pure Text2SQL with semi-structured data (e.g., comma-separated genres in IMDB). SQLite lacks native string-splitting, forcing fragile workarounds. All GPT-4.1 Text2SQL runs failed with runtime errors. TEXT2SQLCODE approaches succeeded by using SQL for bulk retrieval and Python for splitting and aggregation — leveraging each language's strengths for robust, correct results.

Example 2: *Find the TV shows where the absolute rating difference between the first and last episode is the highest. Each season of the TV show can be considered as a separate show.*

This example shows the efficiency of TEXT2SQLCODE for complex per-group logic. This question caused pure Text2SQL runs with GPT-o3-mini to stall for 20+ minutes due to SQL inefficiencies. Our methods split the work: SQL fetched episode data, while Python grouped seasons, found boundary episodes, and computed differences. This division leveraged each language's strengths, produced the correct output, and completed in under three minutes—demonstrating how decomposition scales and performs better than SQL alone.

TEXT2SQLCODE Errors: We manually inspect the errors made by TEXT2SQLCODE$_{multi}$ and TEXT2SQLCODE$_{single}$ systems. We find that the majority of errors are logical mistakes in implementing code/SQL for the user question. Examples include missing clauses (question asks for movies, and query forgets to filter on `titletype=movie`), inaccurate null processing (question requires aggregation per season, and code considers `null` season as a separate season), and missed steps (question requires a final count and answer outputs a list, or question requires a final aggregate, whereas answer outputs it per season). Another error type occurs due to inaccurate decomposition – for example, the question asks for a player's name, but the model misses retaining this information in its Text2SQL prompt. So, the code lacks name data, and can at best output player IDs, leading to execution errors. Other less frequent errors include answers with additional information, or in wrong format/units, and Text2SQL prompts dependent on each other. In most cases, we find that broadly the solution contains some of the correct elements of the query, but includes subtle errors, loses context or performs reasoning errors which ultimately lead to an incorrect answer.

We present additional analysis in Appendix B.

7 Conclusions

We introduce STARQA, the first public human-created dataset for QA over structured data aimed at measuring the abilities of models to perform complex analytical reasoning. In dataset construction, we focused on several specific categories of reasoning such as statistical operations and analysis, time-series analysis, complex conditional logic, multi-criteria filtering, using commonsense knowledge and performing scenario understanding. Through our experiments, we demonstrated that this dataset is challenging for current state-of-the-art LLMs, with the best performance reaching up to 48.3%. We experimented with generating SQL code alone, and also proposed to decompose the solution into using SQL and Python code, where the code can be simpler and more expressive and showed that the latter approach produces generally better results, especially when invoked only if necessary.

Future work can further explore improving the performance of models on this data set by leveraging the idea of task decomposition and potentially combining this into agentic workflows, that can plan, examine and correct their mistakes and invoke other tools. The STARQA itself could be extended to include even more types of analytical reasoning and cover more domains, in order to enable study and methods for cross-domain training.

Limitations

Our dataset and methods are evaluated on three underlying databases that we consider expose several challenges related to complex analytical reasoning.

We acknowledge these do not represent the entire complexity of all databases, although we consider our dataset as a solid first step to explore other data bases. Further, we limit our study to queries and dataset in English, hence we did not test the generalizability to other languages and multi-lingual models, and leave this to future work. Our prompts for testing LLMs are constructed with best prompting strategies from the literature, however there is more scope to perform prompt optimization and tuning in order to achieve the best results. All of our experiments are relevant for SQL databases, but, in principle, the ideas herein should be equally suited to complex reasoning questions on data sources such as a knowledge graph or a semi-structured table.

Acknowledgements

We acknowledge Ella Hoffman-Coyle, and Shuyi Wang for their early discussions on the project. We also thank Danna Zheng for sharing their evaluation code with us. Most of the work was done when Mausam was on a full-time sabbatical at Bloomberg. For the remaining work, Mausam acknowledges support from IBM and Verisk grants to IIT Delhi.

References

Daman Arora, Himanshu Singh, and Mausam. 2023. Have LLMs advanced enough? a challenging problem solving benchmark for large language models. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 7527–7543, Singapore. Association for Computational Linguistics.

Asim Biswal, Liana Patel, Siddarth Jha, Amog Kamsetty, Shu Liu, Joseph E. Gonzalez, Carlos Guestrin, and Matei Zaharia. 2024. Text2sql is not enough: Unifying AI and databases with TAG. *CoRR*, abs/2408.14717.

Avik Dutta, Mukul Singh, Gust Verbruggen, Sumit Gulwani, and Vu Le. 2024. RAR: retrieval-augmented retrieval for code generation in low resource languages. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing, EMNLP 2024, Miami, FL, USA, November 12-16, 2024*, pages 21506–21515. Association for Computational Linguistics.

Prakhar Gurawa and Anjali Dharmik. 2025. Balancing content size in rag-text2sql system. *Preprint*, arXiv:2502.15723.

George Katsogiannis-Meimarakis and Georgia Koutrika. 2023. A survey on deep learning approaches for text-to-sql. *VLDB J.*, 32(4):905–936.

Rahul Kumar, Amar Raja Dibbu, Shrutendra Harsola, Vignesh Subrahmaniam, and Ashutosh Modi. 2024. BookSQL: A large scale text-to-SQL dataset for accounting domain. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pages 497–516, Mexico City, Mexico. Association for Computational Linguistics.

Fangyu Lei, Jixuan Chen, Yuxiao Ye, Ruisheng Cao, Dongchan Shin, Hongjin Su, Zhaoqing Suo, Hongcheng Gao, Wenjing Hu, Pengcheng Yin, and 1 others. 2024. Spider 2.0: Evaluating language models on real-world enterprise text-to-sql workflows. *arXiv preprint arXiv:2411.07763*.

Alexander Hanbo Li, Patrick Ng, Peng Xu, Henghui Zhu, Zhiguo Wang, and Bing Xiang. 2021. Dual reader-parser on hybrid textual and tabular evidence for open domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4078–4088, Online. Association for Computational Linguistics.

Fei Li and H. V. Jagadish. 2014. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84.

Terrance Liu, Shuyi Wang, Daniel Preotiuc-Pietro, Yash Chandarana, and Chirag Gupta. 2025. Calibrating llms for text-to-sql parsing by leveraging sub-clause frequencies. *Preprint*, arXiv:2505.23804.

Xinyu Liu, Shuyu Shen, Boyan Li, Peixian Ma, Runzhi Jiang, Yuyu Luo, Yuxin Zhang, Ju Fan, Guoliang Li, and Nan Tang. 2024. A survey of NL2SQL with large language models: Where are we, and where are we going? *CoRR*, abs/2408.05109.

Karime Maamari and Amine Mhedhbi. 2024. End-to-end text-to-sql generation within an analytics insight engine. *CoRR*, abs/2406.12104.

Makbule Gulcin Ozsoy, Leila Messallem, Jon Besga, and Gianandrea Minneci. 2025. Text2Cypher: Bridging natural language and graph databases. In *Proceedings of the Workshop on Generative AI and Knowledge Graphs (GenAIK)*, pages 100–108, Abu Dhabi, UAE. International Committee on Computational Linguistics.

Mayur Patidar, Riya Sawhney, Avinash Kumar Singh, Biswajit Chatterjee, Mausam, and Indrajit Bhattacharya. 2024. Few-shot transfer learning for knowledge base question answering: Fusing supervised models with in-context learning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers), ACL 2024, Bangkok, Thailand, August 11-16, 2024*, pages

9147–9165. Association for Computational Linguistics.

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 36: Annual Conference on Neural Information Processing Systems 2023, NeurIPS 2023, New Orleans, LA, USA, December 10 - 16, 2023*.

PremAI. 2024. State of text2sql 2024. https://blog.premai.io/state-of-text2sql-2024/. Accessed: 2025-05-19.

Jingqing Ruan, Yihong Chen, Bin Zhang, Zhiwei Xu, Tianpeng Bao, Guoqing Du, Shiwei Shi, Hangyu Mao, Xingyu Zeng, and Rui Zhao. 2023. TPTU: task planning and tool usage of large language model-based AI agents. *CoRR*, abs/2308.03427.

Riya Sawhney, Samrat Yadav, Indrajit Bhattacharya, and Mausam. 2025. Iterative repair with weak verifiers for few-shot transfer in KBQA with unanswerability. In *Findings of the Association for Computational Linguistics, ACL 2025, Vienna, Austria, July 27 - August 1, 2025*, pages 24578–24596. Association for Computational Linguistics.

Jaydeep Sen, Chuan Lei, Abdul Quamar, Fatma Ozcan, Vasilis Efthymiou, Ayushi Dalmia, Greg Stager, Ashish Mittal, Diptikalyan Saha, and Karthik Sankaranarayanan. 2020. ATHENA++: Natural language querying for complex nested sql queries. *Proc. VLDB Endow.*, 13(11):2747–2759.

Liang Shi, Zhengju Tang, and Zhi Yang. 2024. A survey on employing large language models for text-to-sql tasks. *CoRR*, abs/2407.15186.

StackOverflow. 2023. StackOverflow developer survey 2023. https://survey.stackoverflow.co/2023/. Accessed: 2025-05-19.

Elias Stengel-Eskin and Benjamin Van Durme. 2023. Calibrated interpretation: Confidence estimation in semantic parsing. *Transactions of the Association for Computational Linguistics*, 11:1213–1231.

Guanghu Sui, Zhishuai Li, Ziyue Li, Sun Yang, Jingqing Ruan, Hangyu Mao, and Rui Zhao. 2023. Reboost large language model-based text-to-sql, text-to-python, and text-to-function - with real applications in traffic domain. *CoRR*, abs/2310.18752.

Ruoxi Sun, Sercan Ö. Arik, Hootan Nakhost, Hanjun Dai, Rajarishi Sinha, Pengcheng Yin, and Tomas Pfister. 2023. Sql-palm: Improved large language model adaptation for text-to-sql. *CoRR*, abs/2306.00739.

Chang-Yu Tai, Ziru Chen, Tianshu Zhang, Xiang Deng, and Huan Sun. 2023. Exploring chain of thought style prompting for text-to-SQL. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pages 5376–5393, Singapore. Association for Computational Linguistics.

Bing Wang, Changyu Ren, Jian Yang, Xinnian Liang, Jiaqi Bai, LinZheng Chai, Zhao Yan, Qian-Wen Zhang, Di Yin, Xing Sun, and Zhoujun Li. 2025a. MAC-SQL: A multi-agent collaborative framework for text-to-SQL. In *Proceedings of the 31st International Conference on Computational Linguistics*, pages 540–557, Abu Dhabi, UAE. Association for Computational Linguistics.

Jun Wang, Patrick Ng, Alexander Hanbo Li, Jiarong Jiang, Zhiguo Wang, Bing Xiang, Ramesh Nallapati, and Sudipta Sengupta. 2022. Improving text-to-SQL semantic parsing with fine-grained query understanding. In *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, pages 306–312, Abu Dhabi, UAE. Association for Computational Linguistics.

Tianshu Wang, Xiaoyang Chen, Hongyu Lin, Xianpei Han, Le Sun, Hao Wang, and Zhenyu Zeng. 2025b. Dbcopilot: Natural language querying over massive databases via schema routing. In *Proceedings 28th International Conference on Extending Database Technology, EDBT 2025, Barcelona, Spain, March 25-28, 2025*, pages 707–721. OpenProceedings.org.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc V. Le, Ed H. Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. 2023. Self-consistency improves chain of thought reasoning in language models. In *The Eleventh International Conference on Learning Representations, ICLR 2023, Kigali, Rwanda, May 1-5, 2023*. OpenReview.net.

Edward C. Williams, Nakul Gopalan, Mina Rhee, and Stefanie Tellex. 2018. Learning to parse natural language to grounded reward functions with weak supervision. In *2018 IEEE International Conference on Robotics and Automation, ICRA 2018, Brisbane, Australia, May 21-25, 2018*, pages 1–7. IEEE.

Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, Zilin Zhang, and Dragomir R. 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, Brussels, Belgium, October 31 - November 4, 2018*, pages 3911–3921. Association for Computational Linguistics.

Daoguang Zan, Bei Chen, Fengji Zhang, Dianjie Lu, Bingchao Wu, Bei Guan, Wang Yongji, and Jian-Guang Lou. 2023. Large language models meet NL2Code: A survey. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7443–7464, Toronto, Canada. Association for Computational Linguistics.

Yi Zhang, Jan Deriu, George Katsogiannis-Meimarakis, Catherine Kosten, Georgia Koutrika, and Kurt Stockinger. 2023. Sciencebenchmark: A complex real-world benchmark for evaluating natural language to SQL systems. *CoRR*, abs/2306.04743.

Danna Zheng, Mirella Lapata, and Jeff Pan. 2024. Archer: A human-labeled text-to-SQL dataset with arithmetic, commonsense and hypothetical reasoning. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 94–111, St. Julian's, Malta. Association for Computational Linguistics.

Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2sql: Generating structured queries from natural language using reinforcement learning. *CoRR*, abs/1709.00103.

A STARQA Categorization Results

Model	Method	Stats	Non-entries in DB	Nested Joins	Strings	Agg Analytics	Complex Column	Temporal Reasoning	Complex Filtering	Time Series	Scenario	Common sense
Claude 3.7	Text2SQL	42.9	15.1	38.2	41.0	27.9	25.7	20.7	38.9	23.9	41.1	17.2
	Text2SQLCode$_{single}$	41.6	**18.4**	35.1	**47.8**	29.5	22.3	22.0	41.3	**36.8**	35.6	14.9
	Text2SQLCode$_{multi}$	44.3	16.1	41.3	47.5	**37.1**	**35.0**	37.9	40.8	35.6	36.6	**18.7**
	Hybrid$_{single}$	43.3	16.7	41.6	46.1	28.7	22.7	40.3	**47.4**	34.4	**43.1**	12.4
	Hybrid$_{multi}$	**43.8**	14.4	**43.5**	44.4	32.0	25.4	**43.6**	46.4	35.6	38.5	16.1
GPT 4.1	Text2SQL	29.3	19.6	34.6	39.3	36.1	14.2	26.1	34.9	36.4	41.7	12.4
	Text2SQLCode$_{single}$	35.9	24.7	35.6	52.6	36.2	**36.5**	29.5	37.0	46.1	37.9	**22.4**
	Text2SQLCode$_{multi}$	**38.6**	21.3	37.7	44.4	39.2	24.4	31.1	40.9	33.5	36.7	11.2
	Hybrid$_{single}$	34.9	**25.3**	41.8	**49.1**	**41.2**	28.1	**33.3**	42.9	**56.6**	**56.9**	18.7
	Hybrid$_{multi}$	34.4	22.4	**42.3**	47.6	40.9	20.3	33.7	**43.7**	44.9	50.0	13.7
GPT o1	Text2SQL	34.0	15.6	35.1	42.3	35.5	15.8	21.7	44.4	47.5	39.4	23.6
	Text2SQLCode$_{single}$	**49.1**	**24.2**	**45.0**	**65.1**	40.1	**38.0**	**38.3**	49.8	50.4	**42.7**	26.1
	Text2SQLCode$_{multi}$	45.0	17.9	35.4	50.9	34.4	32.8	18.7	42.8	34.6	32.0	31.0
	Hybrid$_{single}$	44.7	19.0	44.9	56.0	**43.0**	27.8	30.9	**52.2**	**58.0**	42.6	**32.4**
	Hybrid$_{multi}$	43.5	19.0	41.9	48.2	41.0	25.0	26.7	46.3	54.5	40.8	31.0
GPT o3-mini	Text2SQL	39.6	17.3	35.3	46.1	34.6	36.2	33.1	48.2	50.7	30.6	26.1
	Text2SQLCode$_{single}$	**43.6**	**27.6**	**41.0**	**55.7**	**37.0**	**38.3**	31.7	50.5	55.6	26.9	26.1
	Text2SQLCode$_{multi}$	42.1	22.4	34.8	50.3	34.5	30.5	24.7	51.0	44.3	**32.0**	24.9
	Hybrid$_{single}$	41.9	20.1	38.4	48.6	32.8	38.0	**37.4**	**55.8**	**55.7**	30.2	29.8
	Hybrid$_{multi}$	42.1	19.0	37.7	48.0	33.6	46.4	34.6	56.0	51.2	29.7	**31.0**
Deepseek V3.1	Text2SQL	26.2	15.5	24.6	37.3	21.2	24.5	22.3	31.6	22.8	25.0	8.7
	Text2SQLCode$_{single}$	30.2	**27.0**	22.5	**47.3**	20.5	15.3	21.4	26.9	24.3	22.7	9.9
	Text2SQLCode$_{multi}$	**45.6**	19.0	**39.6**	46.6	**36.0**	**47.8**	**35.7**	40.5	**40.7**	36.9	**16.8**
	Hybrid$_{single}$	35.9	27.0	31.2	47.2	26.0	38.0	26.8	39.9	31.1	27.3	11.2
	Hybrid$_{multi}$	45.4	20.7	39.2	45.5	35.0	40.3	29.8	**43.4**	40.0	**42.6**	14.9

Table 4: Execution accuracy (average over three runs) over different reasoning categories of STARQA.

Table 4 shows our approaches consistently outperform Text2SQL on questions involving statistical operations, non-entries in DB, string operations, nested joins, and complex filtering. While a few models show a slight degradation for other categories, our models generally perform better. This analysis is suggestive due to the small number of questions in some categories.

B Additional Error Analysis

Text2SQL: Syntactic Error Diversity and Debuggability: Models varied significantly in the types and consistency of their SQL-related errors. Some, like Qwen-2.5 and DeepSeek-v3.1, produced a broad range of failure modes—including unsupported functions, misused columns, and unstable output formatting—indicating weaker alignment with the SQL dialect and greater fragility. In contrast, models like GPT-o3-mini and GPT-o1 generated more uniform and predictable errors, making them easier to debug and more reliable in execution. Models with lower error diversity tended to exhibit stronger syntactic control and were better suited for integration in systems that rely on stable SQL generation.

Text2SQL-SC: Self-Consistency and Majority-Voting Behavior: Models also differed in how consistently they generated predictions across multiple runs. GPT-o3-mini, GPT-o1, and GPT-4.1 produced highly stable outputs, with correct predictions aligning across runs—making them well-suited for majority-voting strategies. In contrast, Qwen-2.5 repeatedly generated the same incorrect outputs, preventing fallback mechanisms (like Text2SQLCode) from activating and leading to consistently poor results. This shows that self-consistency only adds value when the underlying predictions are reliable.

While HYBRID approaches often enhance the robustness of TEXT2SQLCODE, they are not universally better. Problems arise when a model: (1) consistently produces incorrect SQL predictions, causing self-consistency voting to lock in bad outputs, and (2) fails again during Python execution, making recovery impossible even if fallback is triggered.

This was the case with Qwen-2.5, which combined high runtime failure, poor prediction diversity, and low correctness—rendering hybridization ineffective. In contrast, GPT-o3-mini and GPT-o1 demonstrate how hybridization can succeed: their SQL outputs are both reliable and diverse, enabling fallback to Python when necessary and ensuring correctness when it is not.