QATCH: Automatic Evaluation of SQL-Centric Tasks on Proprietary Data

QATCH: Automatic Evaluation of SQL-Centric Tasks on Proprietary Data

SIMONE PAPICCHIO, Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italyand EURECOM, Sophia Antipolis, Alpes | Maritimes, France
PAOLO PAPOTTI, EURECOM, Sophia Antipolis, France
LUCA CAGLIERO, Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy

Tabular Representation Learning (TRL) and Large Language Models (LLMs) have become established for tackling Question Answering (QA) and Semantic Parsing (SP) tasks on tabular data. State-of-the-art models are pre-trained and evaluated on large open-domain datasets. However, the performance on existing QA and SP benchmarks is not necessarily representative of that achieved on proprietary data as the characteristics of the input and the complexity of the posed queries show high variability. To tackle this challenge, our goal is to allow end-users to evaluate TRL and LLM performance on their own proprietary data. We present Query-Aided TRL CHecklist (QATCH), a toolbox to automatically generate a testing checklist tailored to QA and SP. QATCH provides a testing suite highlighting models' strengths and weaknesses on relational tables unseen at training time. The proposed toolbox relies on a SQL query generator that crafts tests of varying types and complexity including, amongst others, tests on null values, projection, selections, joins, group by, and having clauses. QATCH also supports a set of general cross-task performance metrics providing more insights into SQL-related model capabilities than currently used metrics. The empirical results, achieved by state-of-the-art TRL models and LLMs, show substantial performance differences (1) between existing benchmarks and proprietary data, (2) across queries of different complexity.

CCS Concepts: • **Information systems** → **Information systems applications**; *Query languages*; • **Computing methodologies** → **Natural language processing**;

Additional Key Words and Phrases: Tabular Representation Learning, Semantic Parsing, Text2SQL, Table Question Answering, Large Language Models, Query Generation

ACM Reference format:
Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2025. QATCH: Automatic Evaluation of SQL-Centric Tasks on Proprietary Data. *ACM Trans. Intell. Syst. Technol.* 16, 2, Article 45 (April 2025), 26 pages.
https://doi.org/10.1145/3712704

1 Introduction

Tabular Representation Learning (TRL) models have emerged as powerful tools for solving SQL-centric tasks such as Table **Question Answering (QA)**[1] and **Semantic Parsing (SP)** [1].

[1]For the sake of brevity, we will denote Table QA by QA whenever it is clear from the context.

Authors' Contact Information: Simone Papicchio, Politecnico di Torino, Torino, Italy and EURECOM, Sophia Antipolis, Alpes-Maritimes, France; e-mail: simone.papicchio@polito.it; Paolo Papotti, EURECOM, Sophia Antipolis, France; e-mail: ; Luca Cagliero (corresponding author), Dipartimento di Automatica e Informatica, Politecnico di Torino, Turin, Italy; e-mail: luca.cagliero@polito.it.

ACM 2157-6912/2025/4-ART45
https://doi.org/10.1145/3712704

Table 1. Example of Proprietary Table

Order Id	Customer Name	State	Company Name	#Items per Order	Total Amount
O1	John Smith	CA	Nd Inc.	23	28,290
O2	Jane Doe	NY	Al Inc.	14	17,220
O3	Michael Lee	CA	Ap Inc.	32	Null
O4	Emily Jones	WA	Nd Inc.	32	39,360
O5	David Lee	FL	Mt Inc.	12	14,760

The use of pre-trained TRL models in enterprise scenarios is established due to their limited cost. Recently, **Large Language Models (LLMs)** have become popular for tackling QA and SP tasks on tabular data [5]. However, selecting the appropriate model to use on domain-specific, proprietary data can be challenging. Each model is suitable for tables with different characteristics, various downstream tasks, and is trained with specific pre-training strategies and datasets.

Existing open source benchmarks (e.g., Spider [44]) are potentially not representative of the schema- and instance-level patterns that can be observed in proprietary data. In a corporate setting, there are at least three scenarios where a given model needs to be evaluated on proprietary datasets:

(1) *Comparison*: Compare TRL models and LLMs fine-tuned on private examples to see which performs best.
(2) *Validation*: Test every candidate model in enterprise use cases to verify whether the output quality meets the requirements.
(3) *Maintenance*: Recalibrate fine-tuned models to avoid data shifting. Continuous evaluation of new test data is necessary to determine when models fail (and why).

Enterprise end-users often evaluate TRL models and LLMs using carefully designed test sets, typically in a zero-shot scenario. However, writing *ad hoc* tests for SQL-centric tasks like QA and SP is extremely time-consuming, as they involve a mix of **Natural Language (NL)** and SQL declarations posed on large, domain-specific tables.

Let us consider the example of proprietary Table 1. It collects information about each customer's order, including company name, number of requested items, and total amount. To evaluate the performance of TRL models and LLMs in a business setting, enterprise end-users should create customized tests based on their typical SQL queries. It is worth including in the tests a comprehensive set of schema- and instance-level SQL patterns commonly faced during enterprise activities. Test preparation is particularly time-consuming and cannot be substituted by standard open-domain benchmarks (e.g., [20, 23, 44, 45]) as they unlikely reflect the characteristics of enterprise tables and queries [2, 9, 27].

The validation step also requires an end-user effort. It involves matching the model output with the expected result, which is part of the test. However, capturing the reasons behind a possible test failure can be challenging. For example, a QA test on tabular data might fail because the returned tuples or values are written in different order, contain null values, or remove duplicates. Modeling these fine differences requires an understanding of the results' features, intrinsic to the relational model. One of the purposes of this work is to overcome the limitations of existing evaluation procedures [14, 19], which computes the accuracy of the result while ignoring these crucial aspects. The same holds for the SP task, as common practice is to evaluate the produced SQL queries against their results once executed [12, 21, 22].

We propose a new toolbox for testing TRL models and LLMs, namely **Query-Aided TRL CHecklist (QATCH)**, in short. QATCH automatically generates testing checklists based on proprietary data for QA and SP. Checklists are automatically produced by a SQL query generator capable of crafting tests of varying complexity thus ensuring adaptability to different datasets and settings. We intentionally disentangle the model's ability to handle various SQL query patterns from the inherent complexity of NL, which is out of the scope of the present work. Additionally, QATCH introduces a set of cross-task performance metrics, enabling a comprehensive evaluation of the model's performance. These cross-task performance metrics cover aspects related to relational tables, such as the resulting cardinality, the tuple ordering, and the presence of NULL or extra cell values. Compared to popular evaluation metrics like execution accuracy [4], QATCH offers a more insightful assessment of the model's SQL-centric performance.

Our research aims to answer the following questions related to the QA and SP performance achieved by TRL models and LLMs: (1) Given a model, are the performance achieved on proprietary and open-domain benchmark data comparable to each other? The aim is to empirically test the models' capabilities to address SQL-centric tasks on unseen proprietary tables. (2) Given a task (QA or SP), do fine-tuned TRL models achieve performance comparable to LLMs? Here we go through the relation between model performance and complexity. On the one hand, LLMs rely on pre-trained networks that are typically much larger than state-of-the-art TRL models. On the other hand, TRL models are designed to handle tabular data and can be easily fine-tuned. Our objective is to study whether an *ad hoc* fine-tuning of TRL models can fill the gap in model complexity with LLMs. Finally, (3) what are the SQL constructs in the queries corresponding to the NL questions that drive the failure of the models? With a controlled generation process, we can systematically analyze the impact of the different SQL operators on the result quality.

The main article contributions can be summarized as follows:

- *Testing checklist on proprietary data*: We present an evaluation toolbox for TRL models and LLMs that generates and evaluates a testing checklist to assess the model's ability to process proprietary tabular data in two SQL-centric tasks, i.e., SP and QA. Unlike existing open-domain benchmarks [19, 23, 40, 44, 45], QATCH uses enterprise tables in model comparison, validation, and maintenance; testing checklists are generated on demand, providing a precise and up-to-date depiction of the data.
- *Template-based SQL query generator*: We adopt a SQL query generator to craft a testing checklist for two tasks tailored to tabular data (QA and SP). Our test templates have been crafted to assess the performance of models across a range of SQL-centric complexities and are easy to extend according to the user workload.
- *Cross-task evaluation metrics*: We define cross-task performance metrics that capture the nuances in QA and SP models' comparison offering more detailed results than the metrics currently used in literature.
- *Evaluation of performance gap between proprietary and open-domain benchmarks*: We compare the performance of the same model on proprietary and open-domain benchmark data to assess model robustness to unseen tables.
- *Extensive empirical analysis*: We perform extensive experiments on eleven proprietary datasets using six TRL models and three LLMs. We empirically verify whether the fine-tuning of TRL models is sufficient to fill the gap in model complexity with LLMs.
- *Failure analysis*: Our results show that models' failure stems from the datasets at hand: We report significant drop in performance for both LLMs and TRL models when using proprietary data instead of open-domain benchmarks. Our tests also reveal that results degrade with more complex queries, e.g., including constructs such as `NOT NULL`, `DISTINCT`, and `HAVING` negatively affects the quality of the model output.

The remainder of the article is organized as follows. Section 2 reviews the related literature. Section 3 formally introduces the QA and SP tasks on tabular data. Section 4 describes the QATCH toolbox. Section 5 summarizes the main empirical findings, with an in-depth analysis of the results for different SQL categories (see Section 5.2) and comparisons between different models and domains (see Section 5.3). Finally, Section 6 draws conclusions and discusses the limitations of the present work and the future research agenda.

2 Related Work

TRL models leverage the pre-training/fine-tuning paradigm to generate unsupervised vector representations of tabular data suited to address a variety of downstream tasks. This work addresses the automatic testing of TRL models and LLMs for two tasks established for tabular data, i.e., Table QA and SP [1, 2, 8].

Benchmark Datasets. Several benchmarks have been developed to measure model performance on open-domain datasets for both QA [6, 29] and SP [4, 23]. Specifically, the WikiTableQuestions dataset [29] is used to evaluate table comprehension tasks. It includes questions about tables from Wikipedia. The TabFact dataset [6] is focused on fact-checking and contains statements about tables that can be classified as "true" or "false." The Spider dataset and its extensions [4] consist of a corpus of databases, each one containing a set of SQL queries with the associated NL questions to test SP. These datasets have pushed forward the development of TRL models in a wide range of tasks. A major drawback of existing benchmarks is that they are built upon a fixed set of pre-determined examples and lack the ability to generate new tests for different domains. Hence, they are neither versatile nor suited to provide comprehensive evaluations.

Template-Based Benchmarks. Template-based generation of tests is a common practice in SP benchmarking. One approach consists of generating SQL queries first and then produce variants of the NL question using different models [15, 42]. An alternative strategy is to generate NL questions first and then apply linguistic variations to augment the number of test samples [25, 41]. As a drawback, these approaches fall short when neither the SQL code nor the NL questions are available.

Evaluation of QA and SP Models. Tabular QA benchmarking commonly relies on the established execution accuracy metrics [16, 18, 24], whose goal is to verify whether the generated values match the correct answer. Similar metrics are also used for SP benchmarking because they are known to be more reliable than comparing SQL queries directly [12, 21, 31]. However, execution accuracy metrics fail to capture finer aspects in the comparison between different models' predictions. For instance, they neglect the schema of the returned tables while exclusively comparing predicted and expected cell values.

Model robustness is particularly challenging while dealing unseen or perturbed samples [36] as test prompts in benchmarks often differ from the samples of a real-world distribution of interest. Common datasets for testing SP robustness are Spider-SYN [10] and Spider-DK [11]. The former tests SP model robustness against synonym substitution by database perturbation. Instead, Spider-DK evaluates model generalization by question perturbation. More recently, DR-Spider [4] includes both types of perturbations. Similar perturbations could be created automatically for any domain with QATCH, i.e., adding *ad hoc* tests in the QATCH toolbox.

Position of Our Work. Our aim is neither to release a new TRL benchmark dataset nor to design larger and more sophisticated templates for SP. Our purpose, instead, is to automatically define testing suites to evaluate the SP and QA performance of TRL models and LLMs on proprietary tables. To the best of our knowledge, this work is the first attempt to address this issue. We also propose *ad hoc* performance metrics suited to evaluate SQL-centric model capabilities, going beyond the classical

execution accuracy. A preliminary version of QATCH has been presented in [27]. Compared to the present work, the early version (1) disregards various levels of complexity of the SQL queries such as the presence of NULL, the negated predicates, the join, group by, and having clauses; (2) covers a more limited number of TRL models and LLMs; (3) provides only a high-level description of the QATCH steps and a less extensive empirical evaluation; (4) lacks of a detailed comparison between fine-tuned TRL models and LLMs.

3 The Evaluation Problem

We first describe the setting of our problem: the evaluation for the tasks of QA and SP on tabular data using TRL models and LLMs. We then state the three main **Research Questions (RQs)** we address in this work. For the sake of simplicity, hereafter, we refer to TRL models and LLMs as **Language Models (LMs)** in short.

3.1 Problem Statement

Given a question Q posed in NL, the purpose of the QA and SP tasks is to retrieve the correct answer R_{SQL}.

The LM, M, encodes the information for table T to generate possible answers. Let SQL_Q be one SQL declaration that retrieves the correct answer R_{SQL} for Q:

$$R_{\text{SQL}} = \textsc{Query}(\text{SQL}_Q, T). \tag{1}$$

To investigate how well LMs can handle SQL queries with join operations involving multiple tables, we will extend our previous definitions to include the encoding of pairs of tables T_1 and T_2 instead of just a single table T.

QA. The QA task entails processing question Q on T with model M, which is fed with both schema and instances of table T, to obtain the corresponding answer R'_{SQL}:

$$R'_{\text{SQL}} = \textsc{QuestionAnswerer}(Q, M). \tag{2}$$

Solving QA on LMs is helpful to end-users who are not proficient in SQL writing but are willing to query tabular data by posing questions in NL.

SP. The goal of SP is to map a question Q to the corresponding SQL declaration SQL_Q leveraging the LM, M, which is fed with schema and (optionally) instances of table T:

$$\text{SQL'}_{\text{Q}} = \textsc{SemanticParser}(\text{Q}, \text{M}). \tag{3}$$

Evaluation. QATCH's goal is to enable the evaluation of the effectiveness of the LMs in solving QA and SP. Specifically, for QA, it compares the model's output R'_{SQL} at the instance level with the expected output R_{SQL}. Similarly, for SP, it compares the outcome produced by the execution of the output query SQL'_Q on T with the expected output R_{SQL}. This process is designed to guarantee that two syntactically different but semantically equivalent queries yield a positive matching score. Note that the evaluation metrics of QATCH exclusively consider the retrieved and expected tuples. Parsing NL or SQL declarations is out of the scope of the present work.

3.2 RQs

In this work, we aim to address the following RQs.

RQ1: Are LMs effective in tackling QA and SP on proprietary tables? As neural models have been trained on open-domain QA and SP datasets, their performance on existing benchmarks is potentially biased by contamination or overfitting [34]. Given an unseen proprietary table, do LMs perform similarly on proprietary data and open-domain benchmark corpora?

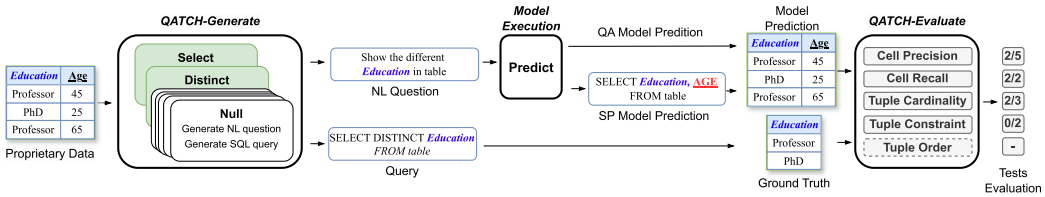

Fig. 1. The QATCH pipeline. Given tables as input, QATCH-GENERATE creates different tests based on SQL templates. The NL question (top part of the generated test) is then executed on a given model. The model's predicted instance is directly collected (QA) or obtained by executing the output script (SP). Given the ground truth query (bottom part of the generated test) and the output of the model at the end, QATCH-EVALUATE calculates five metrics; the *Tuple Order* metric is applied only to tests with the ORDER BY clause.

RQ2: Can fine-tuned tabular LMs match the performance of much larger LLMs in SQL-centric tasks? LLMs feature neural networks that are much larger than TRL models. However, the latter ones are designed to handle tabular data and can be easily fine-tuned. Can fine-tuning fill the gap in model complexity?

RQ3: What are the SQL constructs in the queries corresponding to the NL questions that cause model failure? Intuitively, more complex questions can cause more failure cases in the model predictions. However, what is the impact of the different SQL operators on the result quality?

4 QATCH

QATCH is a testing toolbox to evaluate LMs for proprietary data over QA and SP.

The key steps of QATCH, depicted in Figure 1, are listed below.

— *QATCH-Generate.* This step generates QA and SP tests on tabular data of varying complexity. Given a proprietary table T, it generates a set of SQL declarations SQL_Q, their corresponding NL versions Q, and their expected outputs R_{SQL}.
— *Model Execution.* This step returns the LMs' predictions for the NL questions generated by QATCH-GENERATE.
— *QATCH-Evaluate.* This step highlights strengths and weaknesses of the LMs for QA and SP tasks. It evaluates the models' predictions with their expected outputs R_{SQL} using cross-task quantitative metrics.

4.1 QATCH-Generate

Given a relational table T, QATCH produces test queries consisting of triples $\langle Q, \text{SQL}_Q, R_{\text{SQL}} \rangle$ using a template-based query generator. While the template defines the SQL declaration (e.g., `SELECT` *attribute* `FROM TABLE`), the variables in the templates (e.g., *attribute*) are automatically filled up by the tool according to the values in the schema and active domain on the given T. While the tool is provided by a number of templates that suffice to answer our RQs, user-defined templates of arbitrary complexity can easily be added. We provide an example in Section 5.2.

The main goal of QATCH is to examine LMs' ability to handle various algebraic operations that may be applied directly or indirectly to the relational schema (based on the targeted task). Notice that the focus is neither on the complexity nor the ambiguity of the linguistic expression but rather on analyzing the logical complexity of the question according to the relational data model.

The SQL test suite is comprehensive with various levels of complexity. The main categories are summarized below.

— `PROJECT`: The projection operator selects a subset of attributes in the T's schema.

—DISTINCT: The SQL clause removes duplicated tuples from the output table.
—ORDER BY: The SQL clause sorts the tuples in the output table according to a user-specified criterion.
—SELECT: The selection algebraic operator filters tuples from the T's instance satisfying specific constraints.
—SIMPLE AGGREGATION: The use of SQL functions to compute aggregates over the values of specific T's attributes.
—NATURAL JOIN: The join operator combines the tuples present in two relational table instances based on the values taken by the corresponding attributes.[2]
—GROUP BY: The grouping operator merges tuples sharing the same values for a combination of attributes in T.
—HAVING: The filtering clause shortlists the groups produced by a GROUP BY according to predefined conditions.
—NULL VALUES: The NULL special value to indicate missing or unknown values. We use it in selection conditions.
—NEGATED PREDICATES: The operator enforcing negated Boolean predicates in the selection conditions.

A more detailed analysis of each test category follows.

Project, Distinct, Order by, and Select. To assess the ability of LMs to handle basic algebraic operations, the SQL generator automatically crafts multiple versions of table selections and projections, with or without the DISTINCT clause. Different queries can project all attributes from the schema or subsets of the attributes. QATCH also tests the ability of the models to meet more complex conditions consisting of a mixture of projection and selection operations.

Simple Aggregation. QATCH facilitates test queries with common aggregation functions such as SUM, MIN, MAX, AVG, and COUNT. These functions operate throughout the set of tuples without the GROUP BY clause (see Table 3). The query generator selects attribute subsets based on the type of aggregation function used. For example, categorical attributes are not suitable for the SUM aggregation.

Natural Join. QATCH further assesses the ability of the LMs to handle SQL queries that combine multiple tables using a JOIN clause (see Table 4). These tests focus on the model's capability to join two tables and project specific columns from each table. We intentionally keep the query structure as simple as possible to test the LM's ability to perform joins.

Group by and Having. QATCH also tests the ability of the models to group tuples according to (categorical) attributes (see Table 5) and to filter the generated groups according to the values returned by aggregation functions. The GROUP BY operator is tested with and without the HAVING clause to differentiate between specific models' capabilities.

NULL Values. The presence of NULL values is common in proprietary tables. For this reason, QATCH evaluates the ability of the models to detect and properly handle NULL and NOT NULL values in a table (see Table 6).

Negated Predicates. QATCH assesses the ability to elaborate negated conditions in the WHERE clause (see Table 7).

[2]Unlike all the other categories, this type requires processing a pair of relation tables T_1 and T_2.

Table 2. T Is the Target Table, c_i Denotes an Attribute of T's Schema, op Is a Logical Operator (=, !=, >, <, ≥, ≤), and VAL_i Is an Arbitrary Value for c_i in T's Instance

	PROJECT
Question	Selects all the attributes in the schema of table {T}
Query	SELECT * FROM {T}
Question	Show {$c_1, \ldots, c_n$} in table {T}
Query	SELECT {$c_1, \ldots, c_n$} FROM {T}
	DISTINCT
Question	Show the different {$c_1, \ldots, c_n$} in table {T}
Query	SELECT DISTINCT {$c_1, \ldots, c_n$} FROM {T}
	ORDER BY
Question	Show data for table {T} in {ord} order by {c_i}
Query	SELECT * FROM {T} ORDER BY {c_i} {ord}
	SELECT
Question	Show data of table {t} where {c_i}{op}{val_i}
Query	SELECT * FROM {T} WHERE {c_i} {op} {val_i}

ord indicates the tuple visualization order (ascending or descending).

Table 3. T Is the Target Relational Table

	SIMPLE AGGREGATION
Question	How many different {c_i} are in table {T}?
Query	SELECT COUNT([DISTINCT] {c_i}) FROM {T}
Question	Find the minimum/maximum/average of {n_i} in table {T}.
Query	SELECT MIN/MAX/AVG {n_i} FROM {T}

c_i, n_i are, respectively, a categorical and a numerical attribute of the T schema.

Table 4. c_i Is an Attribute of the T_1 Schema

	NATURAL JOIN
Question	Join the records from table T_1 with table T_2 on c_{ij}
Query	SELECT * FROM T_1 JOIN T_2 ON $T_1.c_{ij} = T_2.c_{ij}$
Question	List c_i and c_j from T_i and T_j where c_{ij} is the same
Query	SELECT c_i, c_j FROM T_1 JOIN T_2 ON $T_1.c_{ij} = T_2.c_{ij}$

c_j is an attribute of the T_2 schema. c_{ij} is an attribute of both the T_1 and the T_2 schema.

4.2 Model Execution

This step uses LMs to generate QA and SP predictions given the NL questions generated by QATCH-Generate. Every prediction of the model is then compared with the ground truth R_{SQL} in the next step QATCH-Evaluate.

Table 5. Group by and Having Tests

	GROUP BY
Question	For each $\{c_i\}$, count the number of rows in table $\{T\}$.
Query	SELECT $\{c_i\}$, COUNT(*) FROM $\{T\}$ GROUP BY $\{c_i\}$
Question	For each $\{c_i\}$, find the {agg} $\{n_j\}$ in table $\{T\}$.
Query	SELECT $\{c_i\}$, {agg}($\{n_j\}$) FROM $\{T\}$ GROUP BY $\{c_i\}$
	HAVING
Question	Find all the $\{c_i\}$ that have {op} μ_{count_i} records in table $\{T\}$.
Query	SELECT $\{c_i\}$ FROM $\{T\}$ GROUP BY $\{c_i\}$ HAVING COUNT(*) {op} μ_{count_i}
Question	List the $\{c_i\}$ which average of $\{n_j\}${op}μ_{avg_i} in table $\{T\}$.
Query	SELECT $\{c_i\}$ FROM $\{T\}$ GROUP BY $\{c_i\}$ HAVING AVG($\{n_j\}$) {op}$\mu_{mean_{ij}}$
Question	List the $\{c_i\}$ which sum of $\{n_j\}${op}μ_{sum_i} in table $\{T\}$.
Query	SELECT $\{c_i\}$ FROM $\{T\}$ GROUP BY $\{c_i\}$ HAVING SUM($\{n_j\}$) {op}$\mu_{sum_{ij}}$

T is the target relational table. c_i, n_i are a categorical and a numerical attribute of the T's schema. μ_{count_i} is the mean of the count row within the group on $\{c_i\}$. $\mu_{avg_{ij}}$ is the mean of the query result of $\{n_j\}$ grouped by $\{c_i\}$. $\mu_{sum_{ij}}$ is the mean of the sum of $\{n_j\}$ grouped by $\{c_i\}$. op is a logical operator $\geq, \leq$. agg is an aggregation (MIN, MAX, AVG, SUM).

Table 6. T Is the Target Relational Table

	NULL VALUES
Question	Count the rows where $\{c_i\}$ is (not) a NULL in table $\{T\}$
Query	SELECT COUNT(*) FROM $\{T\}$ WHERE $\{c_i\}$ IS (NOT) NULL

c_i is an attribute of the T's schema.

Table 7. T Is the Target Relational Table

	NEGATED PREDICATES
Question	Show the data of the table $\{T\}$ where $\{c_i\}$ not equal to val_i
Query	SELECT * FROM $\{T\}$WHERE NOT $\{c_i\}$ = val_i

c_i, n_i are, respectively, a categorical and a numerical attribute of the T's schema. val_i is an arbitrary value for c_i in T's instance.

The QATCH toolkit includes several LMs for both the QA and the SP tasks. For QA, QATCH supports `Tapas` [16], `Tapex` [24], and `Omnitab` [17] TRL models fine-tuned on the WTQ dataset [30]. For SP it supports `GAP` [35], `ResdSQL` [21], and `UnifiedSKG` [43] (the last two used with T5 large [32]). QATCH also supports LLMs: `GPT-3.5-turbo-0613` [26] for both tasks, `Llama2 7B` [38] for QA, and `Llama2-Code` [33] for SP.

4.3 QATCH-Evaluate

For QA the evaluation of models' performance has relied on simplistic accuracy metrics. This evaluation technique, referred to as "execution accuracy," has also been applied to the SP, as it offers a more precise assessment compared to solely comparing SQL queries. However, this approach falls short in capturing the nuanced aspects of evaluating model predictions, as demonstrated in Figure 2. To address these limitations, QATCH uses five performance metrics defined by comparing the model predictions with the ground truth.

Specifically, let I^{pred} and I^{gt} be the predicted and ground truth sets of tuples for query Q, and let $\mathrm{I}_{\mathrm{D}}^{pred}$ and $\mathrm{I}_{\mathrm{D}}^{gt}$ be, respectively, the sets of distinct cell values contained in the prediction and the target. The QATCH metrics are:

Cell Precision. The fraction of table cells $cell_i$ in the output instances that are relevant to the input query. The higher the score, the more predicted elements are in the target.

$$\text{C-Pr} = \frac{|\{cell_i | cell_i \in \mathrm{I}_{\mathrm{D}}^{pred} \cap I_{\mathrm{D}}^{gt}\}|}{|\{cell_i | cell_i \in \mathrm{I}_{\mathrm{D}}^{pred}\}|}.$$

Cell Recall. The fraction of table cells relevant to the input query that are successfully retrieved. The higher the score, the more target cells are included in the prediction.

$$\text{C-Rec} = \frac{|\{cell_i | cell_i \in \mathrm{I}_{\mathrm{D}}^{pred} \cap I_{\mathrm{D}}^{gt}\}|}{|\{cell_i | cell_i \in \mathrm{I}_{\mathrm{D}}^{gt}\}|}.$$

Tuple Constraint. The fraction of ground truth tuples in the query output. It takes value one if the expected and produced output tuples have the same schema, cardinality, and cell values, zero otherwise.

$$\text{T-Cons} = \frac{|\mathrm{I}^{pred} \cap \mathrm{I}^{gt}|}{|\mathrm{I}^{gt}|}.$$

Tuple Cardinality. The ratio of output and ground truth cardinalities. Compared to *Tuple Constraint*, it considers neither the schema nor the cell values. Hence, it should be analyzed jointly with *Cell Precision* and *Cell Recall* to gain insights into models' effectiveness in the QA and SP tasks.

$$\text{T-Card} = \frac{|\mathrm{I}^{pred}|}{|\mathrm{I}^{gt}|}.$$

Tuple Order. The Spearman rank correlation coefficient between the vector representations of the ranked lists of output tuples and ground truth tuples. This non-parametric test is computed only for queries with an `ORDER-BY` clause.

Figure 2 shows examples of performance metrics on a toy dataset. Given the ground truth result (target) with three tuples over two attributes, we report the metric values for two possible model predictions, either from a QA or from executing a query in SP. Notice that:

- *Tuple Constraint*, *Tuple Cardinality*, *Cell Precision*, and *Cell Recall* range between 0 (no matches) and 1 (all matches), while the *Tuple Order* ranges between 0 (opposite rank) and 1 (same rank).
- *Cell Precision* and *Cell Recall* count the distinct matching elements between the ground truth and the prediction.
- The *Tuple Constraint* ignores partial tuple matches (e.g., [*PhD*, 45] is different from [*PhD*, *25*]).
- The combination of *Cell Precision* and *Cell Recall* is equivalent to the Execution Accuracy reported in most QA and SP papers. However, notice that, as shown in Figure 2, the single metrics fall short in several scenarios.

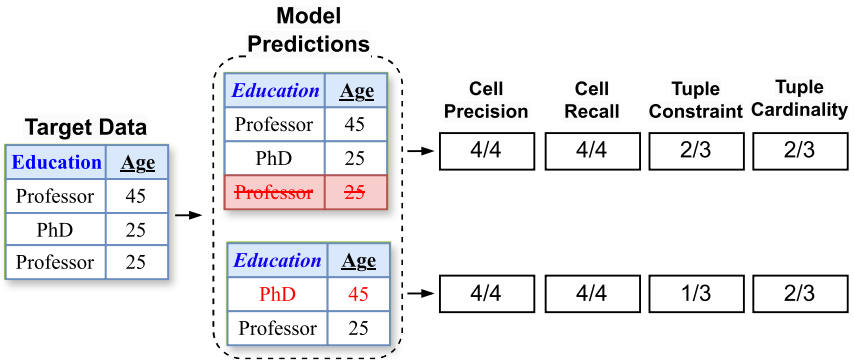

Fig. 2. QATCH evaluates metrics by comparing the model's output (Model Predictions) with the correct answers (Target Data). The first prediction (top) misses the last tuple, whereas the second one (bottom) incorrectly swaps *Education* and *Age* values between two tuples. *Cell Precision* and *Cell Recall* are the metrics most similar to the execution accuracy reported in most QA and SP papers.

To better explain why the *Tuple Constraint* and the *Tuple Cardinality* are necessary, consider the following target values *(a1, b1, c1)*. The first prediction, denoted by "output 1," is composed of two tuples *(a1, b1, c1), (a1, b1, c1)*. Instead, the second prediction, denoted by "output 2," has the following two tuples: *(a1, b1), (c1)*. "Output 1" has cardinality two instead of one, whereas "output 2" has an incorrect schema. In both cases, the *Tuple Constraint* returns zero even if part of the output cells matches. Note also that even if the output cardinality is the same, *Cell Precision* and *Recall* show differences in the returned values.

5 Experimental Evaluation

We analyze the performance of QATCH across eight proprietary datasets and a benchmark collection using six TRL models and three LLMs on the QA and SP tasks. For the LLMs, we consider one representative proprietary model developed by OpenAI Inc. [26] and one representative open source model released by Meta Inc. [38]. Notice that determining the best-performing TRL model or LLM is out of the scope of the present work. Instead, our aim is to answer the RQs stated in Section 3.2. Specifically, the performance comparisons between proprietary and benchmark data (RQ1) are reported in Sections 5.1 and 5.2; the comparative analysis of TRL models' and LLMs' performance (RQ2) are in Sections 5.1, 5.3, and 5.4, and the impact of SQL constructs on performance (RQ3) is discussed in Section 5.2.

Models and Settings. We test the following TRL and LLM models:

—TRL models for QA: `Tapas` [16], `Tapex` [24], `Omnitab` [17]. `Tapas` is a weakly supervised QA model that predicts specific table cells and optionally applies an aggregation operator. `Tapex` takes advantage of pre-training to develop a neural SQL executor using a synthetic set of SQL queries and their execution results. `Omnitab` uses a pre-training method incorporating natural and synthetic data to learn reasoning across multiple tables. We apply Tapas and Tapex fine-tuned on the WTQ [30] dataset.
—TRL models for SP: `ResdSQL` [21], `GAP` [35], `UnifiedSKG` [43]. `ResdSQL` uses a seq2seq architecture with an improved encoding for ranking and a decoding framework aware of the skeleton structure. `UnifideSKG` uses the encoder-decoder model, specifically based on T5.

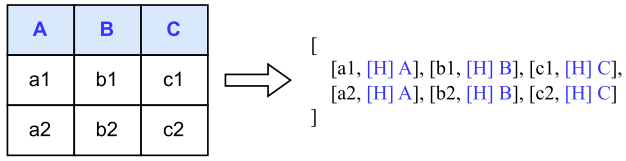

Fig. 3. Toy example of the linearization technique used in this study to process structured data with LLMs.

Both `UnifiedSKG` and `ResdSQL` are employed with the large variant of T5. `GAP` leverages generative models to produce pre-training data, allowing for concurrently learning NL statements and table schemas.

—LLMs models for QA: `GPT-3.5-turbo-06138` [26] and `Llama2-7b` [38]. The two models are tested in the chat version using three examples as a few shots. The system prompt is shown in Listing 1.

—LLMs models for SP: `GPT-3.5-turbo-06138` [26] and `Llama2-Code-7b` [33]. The system prompt is reported in Listing 2. Also for SP, both LLMs are tested with the chat version with three examples as few shots.

```
I want you to act as a question answering model for tabular data.
I will pass you a table with one question.
I want you to only reply with the output of the question executed on the table.
I want you to return the answer in format: list of list (row and columns).
The answer must be complete of all the data from the table.
If an aggregations is present, return only the aggregate values.
Do not write explanations. Do not type commands.
```

Listing 1. QA System Prompt

```
I want you to act as a text to SQL model for tabular data.
I will pass you the schema of the table and one question.
I want you to parse the question into the SQL command.
The SQL command must be executable with the schema of the table.
Do not write explanations. Do not type commands.
```

Listing 2. SP System Prompt

Linearization. We apply a linearization step to tailor tabular data to the requested LLMs' input format. Traditional linearization approaches [37] concatenate table cells using *ad hoc* cell and row delimiters. More sophisticated approaches encode table headings beyond cell content thus incorporating structure-aware indicators [1]. Figure 3 exemplifies the linearization technique used in this study. Each tuple consists of an ordered list of cells, where each cell has a prefix consisting of the corresponding header and the structure-aware indicator `[H]`.

LLMs In-Context Learning. In this study, we employ in-context adaptation techniques to tailor LLMs to QA and SP tasks. In-context learning involves prompting the LLM with training examples and the NL question. To this end, we introduce three different types of training examples in a simple few-shot fashion: (i) Project all table attributes; (ii) Project a single attribute; (iii) Apply an aggregation function. In recent works, various strategies have been used to incorporate more examples [12, 31]. However, we recall that the main goal of this work is not to discover a new SOTA, but rather to evaluate the differences in performance between proprietary and open-domain datasets.

Table 8. Characteristics of the Proprietary Tables

Category	Table Name	#Rows	#Categorical Attrib.	#Numerical Attrib.
ECOMMERCE	Sales-transactions	500k	1	7
	Fitness-trackers	565	6	5
FINANCE	Account-fraud	1M	5	27
	Late-payment	2,466	6	6
MEDICINE	Heart-attack	303	5	9
	Breast-cancer	686	3	9
MISC	Adult-census	32.6k	9	6
	Mushrooms	8.1k	23	0

Table 9. Comparison between Spider [45] and Proprietary Tables

		25% Percentile	50% Percentile	75% Percentile
Spider tables	#tuples	7	12	15
	#attributes	4	5	7
	#cells	30	55	100
Proprietary tables	#tuples	656	5,283	149k
	#attributes	11	12	17
	#cells	7,213	108k	1.4M

Benchmark Datasets. We consider Spider [44] as the reference existing benchmark for both QA and SP. It consists of 200 databases covering 138 domains and a corpus of 6k SQL queries with the corresponding NL question.

Proprietary Datasets. We retrieve proprietary data from the Kaggle platform[3] to maximize the variety of the source domain, dataset cardinality, and dimensionality. Notice that the Kaggle data is available online but lacks the input and target information required for both QA and SP tasks. For these tables, we use QATCH to generate tests for both tasks.

Table 8 reports the main characteristics of the proprietary data. Compared to benchmark data, proprietary tables have both larger cardinality and dimensionality. For instance, in Spider the number of cells per table is 75% lower than in proprietary tables, as reported in the comparative statistics in Table 9. Furthermore, proprietary tables contain a larger number of numerical attributes than the benchmark ones. In the discussion of the results, these differences play a role in understanding the difference between models' performance on proprietary and benchmark data.

To run `Natural Join` tests, we include three additional Kaggle multi-table datasets in the proprietary data collection with characteristics reported in Table 10. The tables we consider have explicit foreign key references. Proprietary dataset links are available on GitHub.[4]

Table Pre-Processing. TRL models and LLMs have constraints on the maximum input tokens, i.e., on the size of the context. To use them in this setting, we limit the length of the input by sampling the tables' attributes and rows using the same random seed for all tables. Going beyond the limit imposed by the context is an open RQ [7].

[3] https://www.kaggle.com/
[4] https://github.com/spapicchio/QATCH

Table 10. Characteristics of Proprietary Multi-Table Datasets

Database Name	#Tables	AVG #Rows	AVG #Cat. Attr.	AVG #Num. Attr.
Basket Dataset	5	631	1	3
Default Risk	6	1,000	6	29
Sepsis System	4	2,193	8	8

QATCH Tests. We use QATCH to generate NL questions and corresponding SQL queries. Then, the SQL queries are executed on the proprietary tables to produce the expected ground truth. Since the Spider benchmark already contains its own NL questions, we explore the following scenarios:

—Use-QATCH-tests-only: Models are tested on QATCH-generated tests from Spider-Dev tables.
—Use-its-own-tests: Models are tested on the questions and queries provided by Spider; to ensure a fair comparison we exclude queries with constructs not included in QATCH, i.e., `LIMIT`, `INTERSECT`, `UNION`, `EXCEPT`, and inner queries.

An analytical comparison between these two scenarios is given in Section 5.4.

How to Interpret SP Metrics. QATCH-Evaluate relies on the cell values for both tasks. In the SP scenario, the process begins with the generation of SQL queries by the models. These queries are then run on the respective tables, leading to the retrieval of predicted cell values. This process enables the comparison of two predictions that are syntactically distinct but semantically equivalent. The QATCH metrics enable us to derive a significant understanding from the SQL query predictions. For instance, a low score in the *Tuple Constraint* could imply the projection of incorrect columns in the SQL query. On the other hand, a low score in the *Tuple Cardinality* could indicate the selection of incorrect rows or the misuse of the DISTINCT operator.

QATCH Execution Time. QATCH-Generate processes the table schema thus only requiring limited computational time (order of seconds). Instead, QATCH-Evaluate's execution time varies according to the size of the table at hand. Comparing prediction and ground truth samples containing 1,000 tuples and 20 attributes takes less than 1 second.

We run our experiments on a machine equipped with a single NVIDIA RTX A6000 48 GB GPU. The inference times varied across different models: `Tapas` and `Tapex` completed their tasks over all tests in a few minutes, while `Llama2` required approximately 4 hours to complete the analysis. Due to the lack of control over OpenAI's infrastructure, we omit the execution times of `GPT-3.5`. Generally speaking, the execution of a single test takes a few seconds. However, this metric's distribution is notably skewed, largely influenced by the variable workload on the server.

5.1 Results' Overview

We compare the performance results produced by the QATCH tests on benchmark and proprietary datasets. Tables 11 and 12 report the results on QA and SP, respectively. The plots in Figure 4 provide an aggregate view of the results by comparing the average performance per metric separately for proprietary and benchmark data. In both tasks, all reported results for Spider-Dev are for the Use-QATCH-tests-only scenario, unless otherwise stated.

Hereafter, we discuss the key findings related to the first two RQs.

RQ1: Are LMs effective in tackling QA and SP on proprietary tables? Figure 4 clearly shows that both TRL models and LLMs underperform with proprietary data compared to open-domain data.

Table 11. QA Results of QATCH Tests

Category	Model QA QA	Cell Precision	Cell Recall	Tuple Cardinality	Tuple Constraint	Tuple Order	Average
			Proprietary data				
ECOMMERCE	Tapas	0.53	0.14	0.28	0.10	0.53	0.32 ± 0.18
	Tapex	0.52	0.19	0.31	0.10	0.59	0.34 ± 0.19
	Omnitab	0.16	0.01	0.30	0.01	0.50	0.20 ± 0.19
	GPT-3.5	**0.61**	**0.39**	**0.37**	**0.24**	**0.63**	**0.45 ± 0.15**
	Llama2	0.11	0.04	0.33	0.01	0.52	0.20 ± 0.19
FINANCE	Tapas	0.59	0.26	0.37	0.14	0.59	0.39 ± 0.18
	Tapex	0.52	0.20	0.37	0.11	0.53	0.35 ± 0.17
	Omnitab	0.15	0.03	0.37	0.02	0.50	0.21 ± 0.19
	GPT-3.5	**0.61**	**0.65**	**0.45**	**0.34**	**0.70**	**0.55 ± 0.13**
	Llama2	0.04	0.02	0.39	0.00	0.53	0.20 ± 0.22
MEDICINE	Tapas	0.53	0.30	0.53	0.20	0.66	0.44 ± 0.17
	Tapex	0.43	0.21	0.33	0.08	0.58	0.33 ± 0.17
	Omnitab	0.29	0.05	**0.55**	0.03	0.53	0.29 ± 0.22
	GPT-3.5	**0.60**	**0.64**	0.32	**0.22**	**0.80**	**0.52 ± 0.21**
	Llama2	0.03	0.01	0.58	0.00	0.53	0.23 ± 0.27
MISC	Tapas	0.54	0.18	0.30	0.09	0.53	0.33 ± 0.18
	Tapex	0.45	0.17	0.36	0.07	0.48	0.31 ± 0.16
	Omnitab	0.40	0.04	0.33	0.01	0.50	0.26 ± 0.20
	GPT-3.5	**0.61**	**0.63**	0.38	**0.25**	**0.80**	**0.53 ± 0.19**
	Llama2	0.23	0.11	**0.39**	0.00	0.53	0.25 ± 0.19
			Benchmark data				
Spider-Dev tables	Tapas	0.75	0.31	0.55	0.15	0.67	0.49 ± 0.22
	Tapex	0.63	0.23	0.30	0.10	0.51	0.35 ± 0.19
	Omnitab	0.39	0.05	0.26	0.03	0.56	0.26 ± 0.2
	GPT-3.5	**0.81**	**0.79**	**0.84**	**0.72**	**0.78**	**0.79 ± 0.04**
	Llama2	0.40	0.36	0.47	0.14	0.57	0.39 ± 0.14

The results of the best performing model per category are written in boldface

The LMs struggle to deliver the same level of performance on unfamiliar proprietary data as they do on more general datasets. The promising average QA performance of `GPT-3.5` on Spider (0.78) is not confirmed by the corresponding results on proprietary data (0.46).

RQ2: Can tabular-specific and fine-tuned TRL models compete with much larger, but more generic, LLMs in terms of QA and SP performance? LLMs and specific TRL models (e.g., `ResdSQL`) have shown to be quite effective in tackling SP, whereas QA is challenging for all models tested. `GPT-3.5` model outperforms all the open source models, both LLMs and TRL, for both tasks thanks to a broader and more robust pre-training. The smaller `Llama2-Code` [33] 7B model is still competitive in SP, while the performance of `Llama2` [38] is significantly worse than that of `GPT-3.5` on QA.

Table 12. SP Results of QATCH Tests

Category	Model SP	Cell Precision	Cell Recall	Tuple Cardinality	Tuple Constraint	Tuple Order	Average
			Proprietary data				
ECOMMERCE	ResdSQL	0.79	0.78	0.76	0.73	**1.00**	0.8 ± 0.09
	GAP	0.68	0.69	0.63	0.59	0.97	0.71 ± 0.13
	UnifiedSKG	0.64	0.63	0.62	0.58	0.98	0.69 ± 0.15
	GPT-3.5	**0.91**	**0.94**	**0.84**	**0.85**	**1.00**	**0.91 ± 0.06**
	Llama2-Code	0.73	0.76	0.67	0.68	**1.00**	0.77 ± 0.12
FINANCE	ResdSQL	0.80	0.78	**0.76**	0.72	**1.00**	0.81 ± 0.09
	GAP	0.58	0.62	0.54	0.52	0.97	0.65 ± 0.17
	UnifiedSKG	0.62	0.62	0.60	0.55	**1.00**	0.68 ± 0.16
	GPT-3.5	**0.81**	**0.86**	0.73	**0.74**	**1.00**	**0.83 ± 0.09**
	Llama2-Code	0.62	0.66	0.58	0.56	**1.00**	0.68 ± 0.16
MEDICINE	ResdSQL	0.72	0.65	0.74	0.58	0.93	0.72 ± 0.12
	GAP	0.50	0.51	0.47	0.43	0.75	0.53 ± 0.11
	UnifiedSKG	0.64	0.63	**0.62**	0.58	0.95	0.68 ± 0.13
	GPT-3.5	**0.74**	**0.84**	**0.62**	**0.62**	**1.00**	**0.76 ± 0.14**
	Llama2-Code	0.67	0.77	0.58	0.60	**1.00**	0.72 ± 0.15
MISC	ResdSQL	0.93	0.90	0.85	0.79	**1.00**	0.89 ± 0.071
	GAP	0.58	0.56	0.52	0.48	**1.00**	0.63 ± 0.19
	UnifiedSKG	0.70	0.65	0.64	0.56	0.98	0.71 ± 0.14
	GPT-3.5	**0.94**	**0.96**	**0.88**	**0.88**	**1.00**	**0.93 ± 0.04**
	Llama2-Code	0.83	0.86	0.83	0.82	0.82	0.83 ± 0.01
			Benchmark data				
Spider-Dev tables	ResdSQL	0.90	0.86	0.94	0.81	0.95	0.89 ± 0.05
	GAP	0.71	0.68	0.70	0.57	0.81	0.69 ± 0.07
	UnifiedSKG	0.70	0.69	0.74	0.64	0.88	0.73 ± 0.08
	GPT-3.5	**0.97**	**0.97**	**0.98**	**0.97**	**1.00**	**0.98 ± 0.01**
	Llama2-Code	0.92	0.92	0.93	0.91	**1.00**	0.94 ± 0.03

The results of the best performing model per category are written in boldface

More Insights into QA Results. `GPT-3.5` achieves acceptable metric scores on QATCH's tests except for *Tuple Constraint*. Limited preservation of intra-tuple value relationships, despite high *Cell Precision* and *Cell Recall*, caused the issue.[5] `Tapas` and `Llama2` are partially competitive against `GPT-3.5` in terms of *Cell Precision*, but show relevant performance drops for all the other metrics. Notably, *Cell Recall* scores are particularly low for all models, even on Spider data, highlighting TRL models' inability to retrieve all requested data. We observe that in several cases the low recall is due to the models returning less attributes in the results.

The performance gap between the benchmark and proprietary data is significant across all metrics. The test results on the existing Spider benchmark data have shown to be not representative enough of all the complex patterns observed in the proprietary data. This is particularly evident for *Tuple Cardinality*, where the tests based on `DISTINCT` and `GROUP BY` clauses clearly highlight the limitations of TRL models and LLMs. `GPT-3.5` performance degrades significantly on proprietary data across all the SQL categories, e.g., *Cell Recall* 0.79 in Spider vs. 0.64 on Medicine (best among proprietary data categories).

[5]This insight is not visible merely with execution accuracy supporting the introduction of new metrics in `QATCH-Evaluate`.

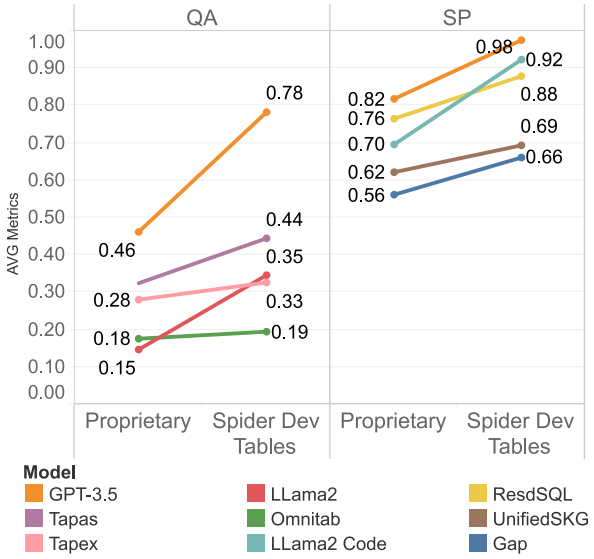

Fig. 4. Average performance comparison between proprietary and benchmark data on QA and SP. A positive slope indicates a rise in the performance with Spider-Dev tables. The higher the gradient, the higher the performance gap.

More Insights into SP Results. SP results confirm the QA trend, with an evident performance gap between the benchmark and the proprietary data. Among the tested models, `GPT-3.5` shows the most robust performance in all datasets. *Tuple Constraints* results are averagely worse than expected, confirming the limitations in handling intra-tuple value relationships for SP as well. More precisely, SP models accurately identify projection columns, exhibiting high *Cell Precision* and *Recall*. Nevertheless, they erroneously include the `DISTINCT` operator in the predicted queries (confirmed by low values of *Tuple Cardinality*). Another problem is the misinterpretation of the `GROUP BY` clauses as select statements.

All TRL models and LLMs have proved to be capable of preserving tuple ordering. Both LLMs are also effective in retrieving the expected cells whereas TRL models other than `ResdSQL` are challenged by QATCH tests on proprietary data, particularly in terms of *Tuple Cardinality* and *Tuple Constraint*. Notably, `UnifiedSKG` and `GAP` show relevant drops in performance on proprietary tables compared to benchmarks.

5.2 Proprietary vs. Benchmark Data: Comparison between Different SQL Query Categories

Figure 5 illustrates the performance gaps between proprietary and the Spider datasets in various categories of SQL queries. For each task, the best performing TRL model and LLM are reported. Specifically, for QA, the comparison is between `GPT-3.5` and `Tapas`, and for SP, it is between `GPT-3.5` and `ResdSQL`.

For the QA task, TRL models show poor performance even for simple SQL query categories (e.g., `PROJECT`, `DISTINCT`). In contrast, `GPT-3.5` generally exhibits superior performance with benchmark data. However, this trend is not consistently observed when applying the same model to proprietary data (on average the distance is of 0.4 points).

For the SP task, `GPT-3.5` is effective in solving simple projections, aggregations, and tuple sorting on both proprietary and benchmark datasets. However, it struggles to enforce the `HAVING`

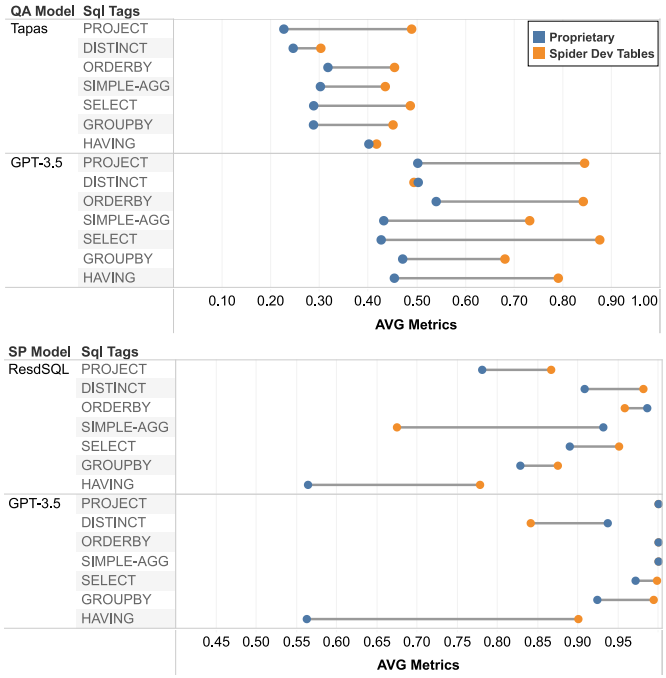

Fig. 5. Performance gaps of the best performing TRL models (top) and LLMs (bottom) between proprietary and Spider tests for every SQL query category. QA on the left and SP on the right.

conditions on tuple groups. In contrast, the TRL models are also challenged by the clauses `SELECT` and `DISTINCT`.

We now analyze the results separately for each of the most significant SQL constructs.

Simple Aggregation. We evaluate the performance of the model over simple aggregations. A description of the tests is given in Table 3. In the context of QA tasks, there is a pronounced disparity in performance compared to SP tasks. In particular, `Tapas` shows robust performance in tests involving MIN or MAX aggregations. However, its effectiveness decreases in scenarios that require the calculation of the mean or count, and its performance almost reduces to zero. `GPT-3.5` outperforms `Tapas`; however, it exhibits a significant lack of generalizability. This is evident in the comparison between datasets; on Spider, `GPT-3.5` scores are nearly around 0.80, but these scores drop to approximately 0.40 when tested on proprietary data. In SP tasks, `GPT-3.5` shows remarkable precision, noting that it does not make incorrect predictions. Moreover, `ResdSQL` exhibits commendable performance, particularly when applied to proprietary data.

Negated Predicates. We test the ability of the models to operate with negated conditions in the WHERE clause (a detailed description of the implementation of the negated select condition can be found in Table 7, whereas the more general select condition is formalized in Table 2). Across both tasks, our analysis does not reveal significant differences between the outcomes of the negated and normal select conditions. The results show a gap in performance between tables including numerical attributes and those consisting of categorical attributes only. For example, the values of the `GPT-3.5` performance metrics are close to zero for table *Sales Transactions*, consisting of seven out of eight numerical attributes.

Grouping and Having Clauses. The robustness of the models is also evaluated in terms of their ability to process NL that requires grouping operations, specifically those involving the `GROUP BY` and `HAVING` clauses. For a detailed account of these tests, see Table 5. For QA, no significant patterns emerge in the performance comparison between `GPT-3.5` and `Tapas`, nor between tests involving `GROUP BY` or `HAVING` clauses. Clauses with `COUNT` in the having conditions generally yield better results. Consistent with previous observations, `GPT-3.5` exhibits a substantial performance disparity between the proprietary and benchmark data.

In the SP task, models particularly struggle with queries involving the `HAVING` clause, displaying a marked performance drop. A key error identified is the models' confusion between the `GROUP BY` and `HAVING` clauses, often mistaking them for select queries. In particular, `GPT-3.5` shows a stark contrast in performance on `GROUP BY` versus `HAVING` tests, the latter seeing a significant reduction in scores. This issue predominantly affects proprietary data, with benchmark data showing consistent results, indicating that the discrepancy may be more related to the nature of the proprietary data than to a model limitation.

Robustness to the Presence of NULL Values. We perform QA and SP tests on proprietary data to verify the abilities of the LMs to handle SQL queries including `NULL` predicates (see Table 6). The results indicate universal failure in all models, including both LLMs and TRLs. The counts returned by the models for the QA task are consistently close to zero, despite the presence of NULL values in the input tables. The prediction of the models for the SP task predominantly contains select statements with the string "null" instead of actually searching for NULL entities.

JOIN Conditions. We assess the ability of SP models to handle `JOIN` conditions (see Table 4). To the best of our knowledge, there is no QA model that can handle input from multiple tables, thereby limiting the scope of our analysis solely to the SP task. To test the SP models, we select three new databases (see Table 10), because the previously described proprietary dataset lacks joinable tables. For proprietary data, we use QATCH-Generate to create NL questions and target outputs. The results are compared with those obtained on the Spider-Dev dataset in the Use-its-own-tests scenario. To ensure a fair comparison, we filter out Spider-Dev queries that do not include any `JOIN` condition. Notice that we neglect the Use-QATCH-tests-only scenario because the current QATCH version is not able to automatically detect columns to join and such piece of information is not explicitly reported in the Spider metadata. As reported in Figure 6, all models, except Llama2-Code, achieve better average performance on Spider-Dev than proprietary data, confirming the trend observed in single-table data. `GPT-3.5` shows the overall best performance on all datasets, whereas `ResdSQL` achieves the worst overall results, since for many NL questions it confuses Join with Select conditions.

We are now ready to answer RQ3.

RQ3: What are the SQL constructs in the queries corresponding to the NL questions that cause model failure? Results in Figures 5 and 6 show that constructs such as `DISTINCT`, `HAVING`, and `JOIN` negatively impact the performance of models in different ways. TRL models clearly suffer from the presence of `JOIN`, while their performance is more robust in the cases where `DISTINCT` must be enforced. The most complex queries, involving `HAVING`, have a significant impact on all models in both tasks.

To conclude the analysis of the impact of query complexity on the quality of the output, we introduce a template for a question requiring a more complicated pattern involving `HAVING` and nesting. Table 13 reports the template and an example based on the *Fitness Trackers* table. The tool can be plugged with any template and automatically produces the tests for the evaluation of the models. In this case, we obtain 88 tests from all proprietary datasets and we focus on `GPT-3.5`

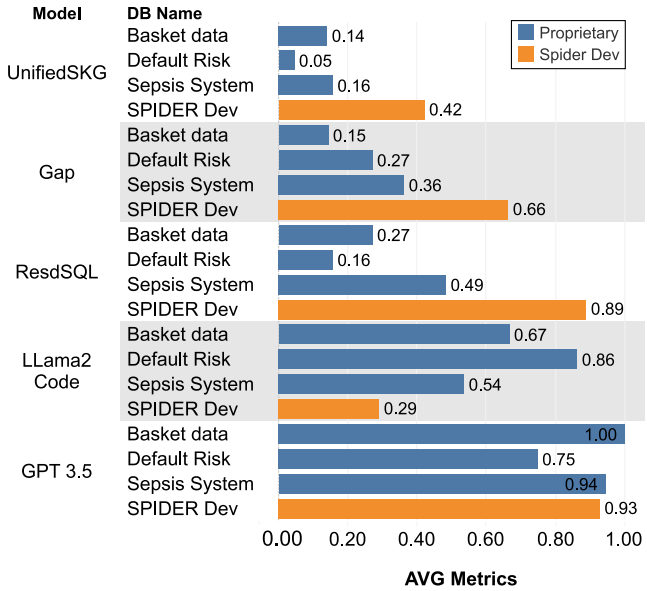

Fig. 6. Results of the Natural Join tests for the SP task: QATCH-GENERATE includes inputs and targets for the proprietary dataset, while the Spider-Dev dataset comprises authentic Spider tests filtered solely on the query with JOIN conditions.

Table 13. Example of a More Complex Template

Many-to-Many	
Question	What are the $\{c_i\}$ with all the $\{c_j\}$ in table $\{T\}$
Query	SELECT $\{c_i\}$ FROM $\{T\}$ GROUP BY $\{c_i\}$ HAVING COUNT(DISTINCT $\{c_j\}$) = (SELECT COUNT(DISTINCT $\{c_j\}$) FROM $\{T\}$);
Many-to-Many Example	
Question	What are the *Brands* with all *Displays* in table *Fitness Trackers*?
Query	SELECT *Brands* FROM *Fitness Trackers* GROUP BY *Brands* HAVING COUNT(DISTINCT *Displays*) = (SELECT COUNT(DISTINCT *Displays*) FROM *Fitness Trackers*);

T is the target table, c_i and c_j denote categorical attributes of T's schema.

as it reports the best performance on all datasets. Results in Table 14 show much lower results compared to those obtained for simpler queries, confirming that higher complexity challenges the best performing model.

5.3 Ablation over the Datasets

Figure 7 shows the performance metrics of various QA models across different proprietary datasets. Specifically, `GPT-3.5` often achieves the best results, whereas `Llama2` and `Omnitab` show consistently worse performance. This disparity underscores the importance of employing the QATCH framework for the complete testing and evaluation of QA models. Notably, `GPT-3.5`'s dominance is not universal. Regarding the datasets related to *Sales Transactions* and *Breast Cancer*, for example, `Tapas` turns out to be the best performing model. `GPT-3.5` is particularly effective in handling

Table 14. Results for the Many-to-Many Template with GPT-3.5 for SP and QA

Category	Cell Precision	Cell Recall	Tuple Cardinality	Tuple Constraint	Average
			SP		
ECOMMERCE	**0.59**	**1.0**	**0.59**	0.0	**0.54 ± 0.36**
FINANCE	0.08	0.53	0.03	0.0	0.16 ± 0.22
MEDICINE	0.37	**1.0**	0.11	0.0	0.37 ± 0.39
MISC	0.29	0.93	0.09	**0.02**	0.33 ± 0.36
			QA		
ECOMMERCE	**0.62**	0.86	**0.52**	**0.29**	**0.57 ± 0.20**
FINANCE	0.31	0.82	0.19	0.04	0.34 ± 0.29
MEDICINE	0.41	0.79	0.19	0.04	0.36 ± 0.28
MISC	0.37	**0.98**	0.27	0.02	0.41 ± 0.35

The results of the best performing category per metric and task are written in boldface

Fig. 7. Average results of QA models on proprietary datasets measured with QATCH metrics.

queries with DISTINCT clauses or with a limited set of output attributes. However, its performance decreases on larger datasets, especially those containing several numerical attributes, e.g., on the *Sales Transactions* dataset GPT-3.5 is challenged by the majority of numerical attributes (seven out of eight).

Figure 8 illustrates the performance metrics of various SP models on different proprietary datasets. Unlike QA, in SP all the LMs show higher variability in performance (e.g., from 0.69 to 1 for GPT-3.5). Overall, GPT-3.5 turns out to be the best performing model, followed by ResdSQL and Llama2-Code. However, GPT-3.5 is again not the best model in all scenarios. For example, on *Account Fraud* and *Breast Cancer* datasets, ResdSQL turns to be more effective than GPT-3.5, mainly due to the better efficacy in jointly handling GROUP BY and HAVING clauses.

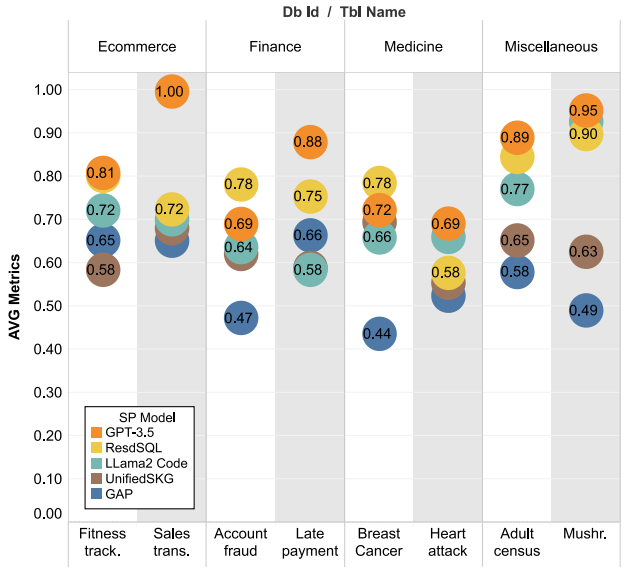

Fig. 8. Average results of SP models on proprietary datasets measured with QATCH metrics.

5.4 Use-QATCH-Tests-Only vs. Use-Its-Own-Tests

This section compares the performance of the LMs achieved in two different scenarios, i.e., when the tests are generated by QATCH with the Spider-Dev's tables (referred to as Use-QATCH-tests-only) or when we use tests from Spider-Dev's dataset (referred to as Use-its-own-tests). In the Use-QATCH-tests-only scenario, both the NL questions and their corresponding targets are automatically created by QATCH-Generate. Conversely, in Use-its-own-tests we use the NL questions and targets available in the Spider-Dev dataset. Therefore, the differences in performance are exclusively due to the variations in the NL questions, since both approaches use the same data source (i.e., Spider-Dev).

Table 15 presents the comparison of the LMs in the QA task for the two approaches. For all models, the Use-QATCH-tests-only scenario yields fairly higher performance scores than Use-its-own-tests, likely due to a higher simplicity of the QATCH's NL questions compared to Spider-Dev questions. Notice that our purpose is not to evaluate model robustness to linguistic challenges, but rather to SQL query patterns.

Table 16 reports similar results for the SP task. `ResdSQL` and `UnifiedSKG` achieve significantly worse results in the Use-QATCH-tests-only scenario compared to the Use-its-own-tests one, possibly due to the specific model pre-training. This confirms the importance of using customized testing toolkits, like QATCH, to evaluate models using their own data.

6 Conclusions and Future Work

In this work, we presented QATCH, a new testing toolkit for TRL models and LLMs on two tasks for tabular data, i.e., QA and SP. One key finding is that models' performance on TRL benchmark datasets is not representative of the expected outcomes on proprietary data due to the inherent variability in data and queries. We also measure the negative impact of increasing query complexity on the models' output. This underpins our research contribution, i.e., a new toolbox that crafts QA and SP tests of varying types and complexity and allows end-users to dig deep into a set of performance metrics based on the comparison of the results against the ground truth output.

Table 15. Comparison of QA Models on Spider-Dev Tables with Different Test Generation: Use-QATCH-tests-only Indicated as QATCH vs. Use-its-own-tests Indicated as SPIDER

QA Model	Test Type	Cell Prec.	Cell Rec.	Tuple Card.	Tuple Cons.	Tuple Ord.	Avg.
GPT-3.5	QATCH	0.80	0.78	0.84	0.72	0.78	**0.78**
	SPIDER	0.62	0.67	0.75	0.58	0.80	0.65
Tapas	QATCH	0.74	0.31	0.55	0.15	0.67	**0.44**
	SPIDER	0.49	0.31	0.44	0.16	0.61	0.35
Tapex	QATCH	0.63	0.24	0.31	0.11	0.51	0.33
	SPIDER	0.44	0.32	0.44	0.18	0.38	**0.34**
Omnitab	QATCH	0.38	0.05	0.27	0.03	0.56	**0.19**
	SPIDER	0.15	0.06	0.45	0.05	0.50	**0.19**
Llama2	QATCH	0.39	0.35	0.47	0.14	0.57	**0.34**
	SPIDER	0.14	0.19	0.36	0.10	0.47	0.21

The best average scores per model are written in boldface

Table 16. Comparison of SP Models on Spider-Dev Tables with Different Test Generation: Use-QATCH-tests-only Indicated as QATCH vs. Use-its-own-tests Indicated as SPIDER

SP Model	Test Type	Cell Prec.	Cell Rec.	Tuple Card.	Tuple Cons.	Tuple Ord.	Avg.
GPT-3.5	QATCH	0.97	0.97	0.98	0.97	1.00	**0.97**
	SPIDER	0.93	0.97	0.98	0.92	0.88	0.95
ResdSQL	QATCH	0.90	0.86	0.94	0.80	0.95	0.88
	SPIDER	0.97	0.97	0.96	0.95	0.97	**0.96**
Llama2 Code	QATCH	0.92	0.92	0.93	0.91	1.00	**0.92**
	SPIDER	0.85	0.87	0.88	0.83	0.74	0.85
UnifiedSKG	QATCH	0.69	0.68	0.74	0.64	0.88	0.69
	SPIDER	0.79	0.81	0.80	0.79	0.40	**0.80**
GAP	QATCH	0.70	0.67	0.69	0.56	0.81	0.65
	SPIDER	0.67	0.66	0.69	0.65	0.23	**0.67**

The best average scores per model are written in boldface

The main *takeaways* from the large set of experiments can be summarized as

—The largest LLM outperforms open source models on both tasks, but shows a relevant performance drop from public benchmark to proprietary data.
—QA appears to be particularly challenging as the state-of-the-art models fail tests including common SQL patterns such as queries with `NULL` values, `DISTINCT`, and `GROUP BY` clauses.
—Despite achieving promising SP results, all models show limitations in handling intra- and inter-tuple relations.
—All models show weaknesses in encoding and querying of tables with numerical attributes.

—QATCH tests are intended to mainly assess the models' capabilities to handle the logical complexity in SQL queries, whereas tests from Spider-Dev's dataset (i.e., Use-its-own-tests) are also influenced by the syntactic complexity of NL questions.

Our *future research agenda* will address the following research lines:

—*Ambiguity in language*: NL questions and table metadata are potentially ambiguous [28, 39]. This yields uncertainty in SQL declarations and calls for testing toolkits tailored to assess model robustness to ambiguity. We intentionally leave ambiguity out of our study to disentangle it from the analysis of SQL patterns.
—*Complexity of SQL query patterns*: Currently supported tests do not involve nested queries, correlations, and tuple constructors. Although current tests are sufficient to surface the limitations of the current tabular models, we envisage the extension of QATCH toward a more comprehensive evaluation of more complex query patterns.
—*Cost benefit analysis*: We plan to perform a cost-benefit analysis to quantify the benefits of using more sophisticated QA and SP models but at much higher complexity.
—*Query efficiency*: Throughout the study, we did not observe relevant differences in the efficiency of the generated SQL scripts across the models. Indeed, the datasets in this work are not large enough to measure sensible differences, but it would be interesting to study differences in query execution times over large databases.
—*Error analysis*: Our metrics surface patterns in the model errors. However, there exist more sophisticated metrics to measure the distance between ground truth and produced instances [13]. Unfortunately, such metrics cannot be directly applied as they come with higher complexity, up to untractable in the presence of NULLs [3].

References

[1] Gilbert Badaro, Mohammed Saeed, and Papotti Paolo. 2023. Transformers for tabular data representation: A survey of models and applications. *Transactions of the Association for Computational Linguistics* 11 (2023), 227–249. DOI: https://doi.org/doi.org/10.1162/tacl_a_00544

[2] Joyce Cahoon, Alexandra Savelieva, Andreas C. Mueller, Avrilia Floratou, Carlo Curino, Hiren Patel, Jordan Henkel, Markus Weimer, Nellie Gustafsson, Richard Wydrowski, et al. 2022. The need for tabular representation learning: An industry perspective. In *NeurIPS 2022 First Table Representation Workshop*, 1–2.

[3] Ashok K. Chandra and Philip M. Merlin. 1977. Optimal implementation of conjunctive queries in relational data bases. In *the 9th Annual ACM Symposium on Theory of Computing (STOC)*. ACM, New York, NY, 77–90. DOI: https://doi.org/10.1145/800105.803397

[4] Shuaichen Chang, Jun Wang, Mingwen Dong, Lin Pan, Henghui Zhu, Alexander Hanbo Li, Wuwei Lan, Sheng Zhang, Jiarong Jiang, Joseph Lilien, et al. 2023. Dr.Spider: A diagnostic evaluation benchmark towards text-to-SQL robustness. In *the 11th International Conference on Learning Representations*, 1–26. Retrieved from https://openreview.net/forum?id=Wc5bmZZU9cy

[5] Yupeng Chang, Xu Wang, Jindong Wang, Yuan Wu, Linyi Yang, Kaijie Zhu, Hao Chen, Xiaoyuan Yi, Cunxiang Wang, Yidong Wang, et al. 2024. A survey on evaluation of large language models. *ACM Transactions on Intelligent Systems and Technology* 15, 3, Article 39 (Mar. 2024), 45 pages. DOI: https://doi.org/10.1145/3641289

[6] Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. arXiv:1909.02164. Retrieved from https://arxiv.org/abs/1909.02164

[7] Giulio Corallo and Paolo Papotti. 2024. FINCH: Prompt-guided key-value cache compression for large language models. *Transactions of the Association for Computational Linguistics* 12 (Nov. 2024), 1517–1532. DOI: https://doi.org/10.1162/tacl_a_00716

[8] Xi Fang, Weijie Xu, Fiona Anting Tan, Jiani Zhang, Ziqing Hu, Yanjun Qi, Scott Nickleach, Diego Socolinsky, Srinivasan Sengamedu, and Christos Faloutsos. 2024. Large language models on tabular data–A survey. arXiv:2402.17944. Retrieved from https://arxiv.org/abs/2402.17944

[9] Avrilia Floratou, Fotis Psallidas, Fuheng Zhao, Shaleen Deep, Gunther Hagleither, Wangda Tan, Joyce Cahoon, Rana Alotaibi, Jordan Henkel, Abhik Singla, et al. 2024. NL2SQL is a solved problem... Not! In *Conference on Innovative Data Systems Research*, 1–8. Retrieved from https://api.semanticscholar.org/CorpusID:266729311

[10] Yujian Gan, Xinyun Chen, Qiuping Huang, Matthew Purver, John R. Woodward, Jinxia Xie, and Pengsheng Huang. 2021. Towards robustness of text-to-SQL models against synonym substitution. In *59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Vol. 1: Long Papers, 2505–2515.

[11] Yujian Gan, Xinyun Chen, and Matthew Purver. 2021. Exploring underexplored limitations of cross-domain text-to-SQL generalization. In *2021 Conference on Empirical Methods in Natural Language Processing*, 8926–8931.

[12] Dawei Gao, Haibin Wang, Yaliang Li, Xiuyu Sun, Yichen Qian, Bolin Ding, and Jingren Zhou. 2023. Text-to-SQL empowered by large language models: A benchmark evaluation. arXiv:2308.15363. Retrieved from https://arxiv.org/abs/2308.15363

[13] Boris Glavic, Giansalvatore Mecca, Renée J. Miller, Paolo Papotti, Donatello Santoro, and Enzo Veltri. 2024. Similarity measures for incomplete database instances. In *27th International Conference on Extending Database Technology (EDBT '24)*. OpenProceedings.org, 461–473. DOI: https://doi.org/10.48786/EDBT.2024.40

[14] Jorge Osés Grijalba, L. Alfonso Urena Lopez, Eugenio Martínez-Cámara, and Jose Camacho-Collados. 2024. Question answering over tabular data with DataBench: A large-scale empirical evaluation of LLMs. In *2024 Joint International Conference on Computational Linguistics, Language Resources and Evaluation (LREC-COLING '24)*, 13471–13488.

[15] Daya Guo, Yibo Sun, Duyu Tang, Nan Duan, Jian Yin, Hong Chi, James Cao, Peng Chen, and Ming Zhou. 2018. Question generation from SQL queries improves neural semantic parsing. In *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, 1597–1607. DOI: https://doi.org/10.18653/v1/D18-1188

[16] Jonathan Herzig, Pawel Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Eisenschlos. 2020. TaPas: Weakly supervised table parsing via pre-training. In *58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 4320–4333. DOI: https://doi.org/10.18653/v1/2020.acl-main.398

[17] Zhengbao Jiang, Yi Mao, Pengcheng He, Graham Neubig, and Weizhu Chen. 2022. OmniTab: Pretraining with natural and synthetic data for few-shot table-based question answering. In *2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Seattle, 932–942. DOI: https://doi.org/10.18653/v1/2022.naacl-main.68

[18] Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. 2024. OpenTab: Advancing large language models as open-domain table reasoners. In *the 12th International Conference on Learning Representations*, 1–17. Retrieved from https://openreview.net/forum?id=Qa0ULgosc9

[19] Sunjun Kweon, Yeonsu Kwon, Seonhee Cho, Yohan Jo, and Edward Choi. 2023. Open-WikiTable: Dataset for open domain question answering with complex reasoning over table. In *Annual Meeting of the Association for Computational Linguistics*, 8285–8297. Retrieved from https://api.semanticscholar.org/CorpusID:258676308

[20] Chia-Hsuan Lee, Oleksandr Polozov, and Matthew Richardson. 2021. KaggleDBQA: Realistic evaluation of text-to-SQL parsers. In *59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing*, Vol. 1: Long Papers, 2261–2273.

[21] Haoyang Li, Jing Zhang, Cuiping Li, and Hong Chen. 2023. RESDSQL: Decoupling schema linking and skeleton parsing for text-to-SQL. In *AAAI Conference on Artificial Intelligence*, Vol. 37, 13067–13075.

[22] Jinyang Li, Binyuan Hui, Reynold Cheng, Bowen Qin, Chenhao Ma, Nan Huo, Fei Huang, Wenyu Du, Luo Si, and Yongbin Li. 2023. Graphix-t5: Mixing pre-trained transformers with graph-aware layers for text-to-SQL parsing. In *AAAI Conference on Artificial Intelligence*, Vol. 37, 13076–13084.

[23] Jinyang Li, Binyuan Hui, Ge Qu, Jiaxi Yang, Binhua Li, Bowen Li, Bailin Wang, Bowen Qin, Ruiying Geng, Nan Huo, et al. 2024. Can LLM already serve as a database interface? A big bench for large-scale database grounded text-to-SQLs. In *37th International Conference on Neural Information Processing Systems (NIPS '23)*. Curran Associates Inc., Red Hook, NY, Article 1835, 28 pages.

[24] Qian Liu, Bei Chen, Jiaqi Guo, Morteza Ziyadi, Zeqi Lin, Weizhu Chen, and Jian-Guang Lou. 2021. TAPEX: Table pre-training via learning a neural SQL executor. In *International Conference on Learning Representations*, 1–19.

[25] Pingchuan Ma and Shuai Wang. 2021. MT-Teql: Evaluating and augmenting neural NLIDB on real-world linguistic and schema variations. *Proceedings of the VLDB Endowment* 15, 3 (Nov. 2021), 569–582. DOI: https://doi.org/10.14778/3494124.3494139

[26] OpenAI. 2023. ChatGPT 3.5. Retrieved from https://openai.com/blog/chatgpt

[27] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2023. QATCH: Benchmarking SQL-centric tasks with table representation learning models on your data. In *37th Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 30898–30917.

[28] Simone Papicchio, Paolo Papotti, and Luca Cagliero. 2024. Evaluating ambiguous questions in semantic parsing. In *2024 IEEE 40th International Conference on Data Engineering Workshops (ICDEW)*, 338–342. DOI: https://doi.org/10.1109/ICDEW61823.2024.00050

[29] Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. In *53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing*. Chengqing Zong and Michael Strube (Eds.), Vol. 1, Association for Computational Linguistics, Beijing, China, 1470–1480. DOI: https://doi.org/10.3115/v1/P15-1142

[30] Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. arXiv:1508.00305. Retrieved from https://arxiv.org/abs/1508.00305

[31] Mohammadreza Pourreza and Davood Rafiei. 2024. DIN-SQL: Decomposed in-context learning of text-to-SQL with self-correction. In *Advances in Neural Information Processing Systems*, Vol. 36, 36339–36348.

[32] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the limits of transfer learning with a unified text-to-text transformer. *Journal of Machine Learning Research* 21, 140 (2020), 1–67. Retrieved from http://jmlr.org/papers/v21/20-074.html

[33] Baptiste Roziere, Jonas Gehring, Fabian Gloeckle, Sten Sootla, Itai Gat, Xiaoqing Ellen Tan, Yossi Adi, Jingyu Liu, Tal Remez, Jérémy Rapin, et al. 2023. Code Llama: Open foundation models for code. arXiv:2308.12950. Retrieved from https://arxiv.org/abs/2308.12950

[34] Oscar Sainz, Jon Campos, Iker García-Ferrero, Julen Etxaniz, Oier Lopez de Lacalle, and Eneko Agirre. 2023. NLP evaluation in trouble: On the need to measure LLM data contamination for each benchmark. In *Findings of the Association for Computational Linguistics (EMNLP '23)*. Association for Computational Linguistics, Singapore, 10776–10787. DOI: https://doi.org/10.18653/v1/2023.findings-emnlp.722

[35] Peng Shi, Patrick Ng, Zhiguo Wang, Henghui Zhu, Alexander Hanbo Li, Jun Wang, Cicero Nogueira dos Santos, and Bing Xiang. 2021. Learning contextual representations for semantic parsing with generation-augmented pre-training. In *AAAI Conference on Artificial Intelligence*, Vol. 35, 13806–13814.

[36] Charlotte Siska, Katerina Marazopoulou, Melissa Ailem, and James Bono. 2024. Examining the robustness of LLM evaluation to the distributional assumptions of benchmarks. In *62nd Annual Meeting of the Association for Computational Linguistics*. Lun-Wei Ku, Andre Martins, and Vivek Srikumar (Eds.), Vol. 1, Association for Computational Linguistics, Bangkok, Thailand, 10406–10421. DOI: https://doi.org/10.18653/v1/2024.acl-long.560

[37] Yuan Sui, Mengyu Zhou, Mingjie Zhou, Shi Han, and Dongmei Zhang. 2024. Table meets LLM: Can large language models understand structured table data? A benchmark and empirical study. In *17th ACM International Conference on Web Search and Data Mining*, 645–654.

[38] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. 2023. Llama 2: Open foundation and fine-tuned chat models. arXiv:2307.09288. Retrieved from https://arxiv.org/abs/2307.09288

[39] Enzo Veltri, Gilbert Badaro, Mohammed Saeed, and Paolo Papotti. 2023. Data ambiguity profiling for the generation of training examples. In *International Conference on Data Engineering (ICDE)*. IEEE, 450–463. DOI: https://doi.org/10.1109/ICDE55515.2023.00041

[40] Denny Vrandečić and Markus Krötzsch. 2014. Wikidata: A free collaborative knowledgebase. *Communications of the ACM* 57, 10 (Sept. 2014), 78–85. DOI: https://doi.org/10.1145/2629489

[41] Nathaniel Weir, Prasetya Utama, Alex Galakatos, Andrew Crotty, Amir Ilkhechi, Shekar Ramaswamy, Rohin Bhushan, Nadja Geisler, Benjamin Hättasch, Steffen Eger, et al. 2020. DBPal: A fully pluggable NL2SQL training pipeline. In *2020 ACM SIGMOD International Conference on Management of Data (SIGMOD '20)*. ACM, New York, NY, 2347–2361. DOI: https://doi.org/10.1145/3318464.3380589

[42] Kun Wu, Lijie Wang, Zhenghua Li, Ao Zhang, Xinyan Xiao, Hua Wu, Min Zhang, and Haifeng Wang. 2021. Data augmentation with hierarchical SQL-to-question generation for cross-domain text-to-SQL parsing. In *2021 Conference on Empirical Methods in Natural Language Processing*. Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (Eds.), Association for Computational Linguistics, Dominican Republic, 8974–8983. DOI: https://doi.org/10.18653/v1/2021.emnlp-main.707

[43] Tianbao Xie, Chen Henry Wu, Peng Shi, Ruiqi Zhong, Torsten Scholak, Michihiro Yasunaga, Chien-Sheng Wu, Ming Zhong, Pengcheng Yin, Sida I. Wang, et al. 2022. UnifiedSKG: Unifying and multi-tasking structured knowledge grounding with text-to-text language models. arXiv:2201.05966. Retrieved from https://arxiv.org/abs/2201.05966

[44] Tao Yu, Rui Zhang, Kai Yang, Michihiro Yasunaga, Dongxu Wang, Zifan Li, James Ma, Irene Li, Qingning Yao, Shanelle Roman, et al. 2018. Spider: A large-scale human-labeled dataset for complex and cross-domain semantic parsing and text-to-SQL task. In *Empirical Methods in Natural Language Processing (EMNLP)*. ACL, 3911–3921. DOI: https://doi.org/10.18653/v1/D18-1425

[45] Victor Zhong, Caiming Xiong, and Richard Socher. 2017. Seq2SQL: Generating structured queries from natural language using reinforcement learning. arXiv:1709.00103. Retrieved from https://arxiv.org/abs/1709.00103

Received 13 August 2024; revised 3 December 2024; accepted 26 December 2024