# Interpretable LLM-based Table Question Answering

**Anonymous authors**
**Paper under double-blind review**

## Abstract

Interpretability in Table Question Answering (Table QA) is critical, especially in high-stakes domains like finance and healthcare. While recent Table QA approaches based on Large Language Models (LLMs) achieve high accuracy, they often produce *ambiguous* explanations of how answers are derived. We propose Plan-of-SQLs (`POS`), a new Table QA method that makes the model's decision-making process interpretable. `POS` decomposes a query into a sequence of atomic steps, each directly translated into an executable SQL command on the table, thereby ensuring that every intermediate result is transparent. Through extensive experiments, we show that: First, `POS` generates the highest-quality explanations among compared methods, which markedly improves the users' ability to simulate and verify the model's decisions. Second, when evaluated on standard Table QA benchmarks (TabFact, WikiTQ, and FetaQA), `POS` achieves QA accuracy that is competitive to existing methods, while also offering greater efficiency—requiring significantly fewer LLM calls and table database queries (up to 25× fewer)—and robust performance on large-sized tables. Finally, we observe high agreement (over 90% in forward simulation) between LLMs and human users when making decisions based on the same explanations, suggesting that LLMs could serve as an effective proxy for humans in evaluating Table QA explanations.

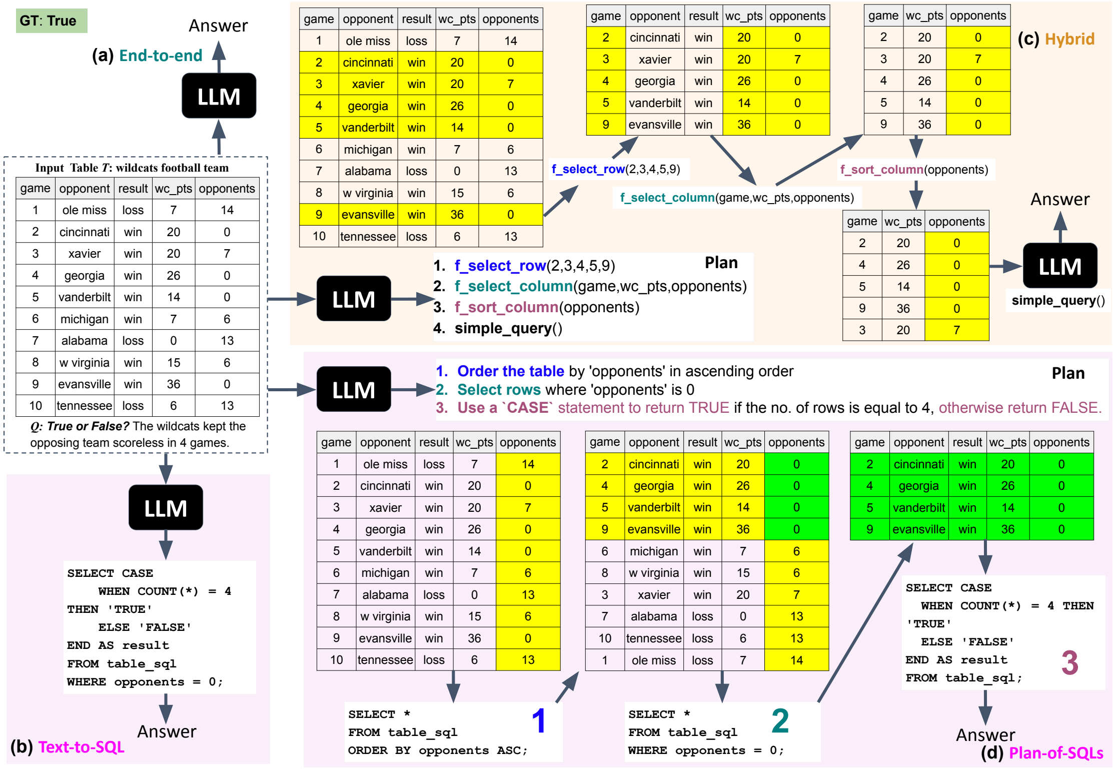
## 1 Introduction

An estimated 38% of office tasks involve working with tables, often using Excel (Richardson, 2022), highlighting the need for advanced tools for tabular data analysis. LLM-powered Table QA models (those in Fig. 1) address this gap by enabling users to quickly extract insights or answer questions for tables, making them invaluable in various industries. For example, financial analysts leverage these models to predict trends from tabular market data (Lo & Ross, 2024). Similarly, medical professionals use them to analyze tabular medical records of patients, facilitating accurate and timely treatment decisions (Bardhan et al., 2022).

However, the value of these systems comes with high risks. Errors in financial decision making have led to catastrophic outcomes, such as the billion-dollar loss Citigroup faced in 2022 (Jane, 2024). In healthcare, the stakes are even fatal, considering a model that misjudges a man's health by overlooking his family history, resulting in his death from cardiac arrest weeks later (Stanford HAI, 2024). These examples underscore the pressing need for interpretability in Table QA to ensure safe and accountable use of AI (Fang et al., 2024).

Despite its importance, interpretability remains an underexplored dimension in Table QA literature. Recent approaches have significantly increased accuracy and often present themselves as *interpretable* solutions (Ye et al., 2023; Cheng et al., 2023; Wang et al., 2024), but this interpretability is unsubstantiated by empirical evidence. In practice, the explanations provided by these models can be unclear. For instance, as shown in Fig. 1**c**, a user cannot discern why certain rows were selected by a function **f_select_row()** or how an operation like **simple_query()** produced the final answer. In other words, current Table QA methods do not adequately explain their reasoning to users.

To address this gap, we propose Plan-of-SQLs (or `POS`), an LLM-based Table QA approach that places interpretability at its core. `POS` decomposes each question into a sequence of *atomic* steps, where each step is a simple sub-query that can be translated into a corresponding SQL command and executed on the table. By design, each transformation is self-contained and limited in scope, forcing the model's decision-making to be broken down into transparent and verifiable steps. This has two key benefits. First, by requiring simple step-by-step table transformations via SQL, we avoid the model arbitrarily pulling in irrelevant data for answering. For example, existing methods that select a large subtable in one shot (Ye et al., 2023; Wang



**Input Table *T*: wildcats football team**

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 1 | ole miss | loss | 7 | 14 |
| 2 | cincinnati | win | 20 | 0 |
| 3 | xavier | win | 20 | 7 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 6 | michigan | win | 7 | 6 |
| 7 | alabama | loss | 0 | 13 |
| 8 | w virginia | win | 15 | 6 |
| 9 | evansville | win | 36 | 0 |
| 10 | tennessee | loss | 6 | 13 |

***Q: True or False?*** The wildcats kept the opposing team scoreless in 4 games.

```
SELECT CASE
    WHEN COUNT(*) = 4
THEN 'TRUE'
    ELSE 'FALSE'
END AS result
FROM table_sql
WHERE opponents = 0;
```

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 1 | ole miss | loss | 7 | 14 |
| 2 | cincinnati | win | 20 | 0 |
| 3 | xavier | win | 20 | 7 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 6 | michigan | win | 7 | 6 |
| 7 | alabama | loss | 0 | 13 |
| 8 | w virginia | win | 15 | 6 |
| 9 | evansville | win | 36 | 0 |
| 10 | tennessee | loss | 6 | 13 |

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 3 | xavier | win | 20 | 7 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |

| game | wc_pts | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 3 | 20 | 7 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |

| game | wc_pts | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |
| 3 | 20 | 7 |

**Plan**

1. f_select_row(2,3,4,5,9)
2. f_select_column(game,wc_pts,opponents)
3. f_sort_column(opponents)
4. simple_query()

**Plan**

1. Order the table by 'opponents' in ascending order
2. Select rows where 'opponents' is 0
3. Use a `CASE` statement to return TRUE if the no. of rows is equal to 4, otherwise return FALSE.

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 1 | ole miss | loss | 7 | 14 |
| 2 | cincinnati | win | 20 | 0 |
| 3 | xavier | win | 20 | 7 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 6 | michigan | win | 7 | 6 |
| 7 | alabama | loss | 0 | 13 |
| 8 | w virginia | win | 15 | 6 |
| 9 | evansville | win | 36 | 0 |
| 10 | tennessee | loss | 6 | 13 |

```
SELECT *
FROM table_sql
ORDER BY opponents ASC;
```

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| 3 | xavier | win | 20 | 7 |
| 7 | alabama | loss | 0 | 13 |
| 10 | tennessee | loss | 6 | 13 |
| 1 | ole miss | loss | 7 | 14 |

```
SELECT *
FROM table_sql
WHERE opponents = 0;
```

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |

```
SELECT CASE
  WHEN COUNT(*) = 4 THEN
'TRUE'
  ELSE 'FALSE'
END AS result
FROM table_sql;
```

Figure 1: **(a)** End-to-End: relies entirely on an LLM to answer the question directly, leaving no room for users to understand the prediction. **(b)** Text-to-SQL: Generates an SQL command to solve the query, requiring domain expertise to understand and becoming unintelligible when the query becomes complex. **(c)** Chain-of-Table or CoTable: performs planning with abstract functions and executes sequentially to arrive at the final answer. However, function arguments are not justified, and the final answer depends on the LLM's opaque reasoning. **(d)** Plan-of-SQLs or `POS` (Ours): plans in natural language, making each step simple and understandable. Each step is then converted into an SQL command, sequentially transforming the input table end-to-end to produce the final answer. We provide a public interface to compare explanations.

et al., 2024) often include spurious entries due to the black-box LLM reasoning (see the irrelevant selection of `row3` in Fig. 1**c**). By contrast, `POS` uses programmatic conditions (e.g. "**WHERE** *opponents* = 0") to ensure only relevant entries are chosen (Step 2 of Fig. 1**d**). Second, `POS` eliminates the opaque answer-generation step commonly found in LLM-based Table QA literature (Ye et al., 2023; Wang et al., 2024; Nahid & Rafiei, 2024b; Zhao et al., 2024) (illustrated in Fig. 1**a**,**c**). Rather than sending the entire table to an LLM, `POS` produces its final answer via a deterministic SQL query execution (Step 3 of Fig. 1**d**), making it entirely clear how the answer is derived from the data.

We thoroughly evaluate the interpretability and performance of `POS`. We compare against existing Table QA methods that provide explanations, including Text-to-SQL (Rajkumar et al., 2022), DATER (Ye et al., 2023), Chain-of-Table (CoTable)(Wang et al., 2024), and self-explaining (Madsen et al., 2024). Our interpretability evaluation spans three benchmarks—(1) explanation quality preference ranking (Ramaswamy et al., 2023; Yang et al., 2024), (2) forward simulation of the model's behavior (Doshi-Velez & Kim, 2017; Hase & Bansal, 2020; Chen et al., 2022; Mills et al., 2023), and (3) model prediction verification (Nguyen et al., 2021; Taesiri et al., 2022)—using both human participants and LLM-based judges. In all settings, `POS` consistently yields the best results, showing that its explanations help users (human or AI) simulate and verify the model output far more effectively than the existing interpretable competitors. Furthermore, when tested on standard datasets (TabFact (Chen et al., 2020), WikiTQ (Pasupat & Liang, 2015), and FeTaQA (Nan et al., 2022)), `POS` achieves accuracy on par with existing methods, while requiring drastically fewer LLM calls and database queries. Notably, `POS` scales robustly to large tables, where current methods Cheng et al. (2023); Ye et al. (2023); Wang et al. (2024) often struggle. Finally, we observe a high agreement between human evaluators and LLM-based evaluators (up to 90%) in forward simulation, suggesting that LLM-based proxies can reliably stand in for human judgment. In summary, our contributions are as follows.

- We introduce `POS`, a new Table QA method that is designed for interpretability. We carry out a thorough study of explanation effectiveness, and show that our explanations substantially improve users' understanding[1] of the model's decision-making over existing interpretable methods (see Sec. 4.1.3).
- Compared to existing Table QA approaches, `POS` is more robust on large tables and far more efficient in its use of LLM calls and database queries (up to 25×), while still achieving competitive QA accuracy (see Sec. 4.2).
- Our experiments reveal high agreement between human users and LLM judges in evaluating Table QA explanations (over 90% agreement in forward simulation). This suggests that LLMs could effectively proxy for human evaluators in evaluating Table QA explanations (see Sec. 4.1.5).

## 2 Related Work

### 2.1 Atomic Table Transformations for Table QA

LLM-based Table QA models have improved performance by decomposing complex input queries into smaller problems (Ye et al., 2023; Nahid & Rafiei, 2024b; Zhao et al., 2024) or by employing step-by-step reasoning (Wang et al., 2024; Wu & Feng, 2024; Abhyankar et al., 2024). However, these approaches often rely on *highly complex* table transformations—for instance, selecting a **query-relevant subtable** from a large input table via the opaque reasoning of LLMs (Ye et al., 2023; Nahid & Rafiei, 2024b; Wu & Feng, 2024; Abhyankar et al., 2024)—which can lead to errors in retrieving the correct data (we detail this problem in Appendix G). Often, the resulting sub-tables contain entries that are either irrelevant or logically incorrect due to the long-context challenges inherent in LLM reasoning.

In contrast, `POS` constrains a transformation of the table to an atomic SQL operation, that is, a single clause with at most one condition and one variable (e.g. "Select rows where `opponents` = 0"). This design has two main advantages. First, it improves accuracy and comprehensibility: because each step is minimal and focused, there is less room for LLMs to make mistakes, and the operation is easy for users to digest. Second, it enables fine-grained attribution: by executing each step with an SQL query, we can pinpoint exactly which input cells were used at that step, yielding a detailed trace of how the final answer is derived.

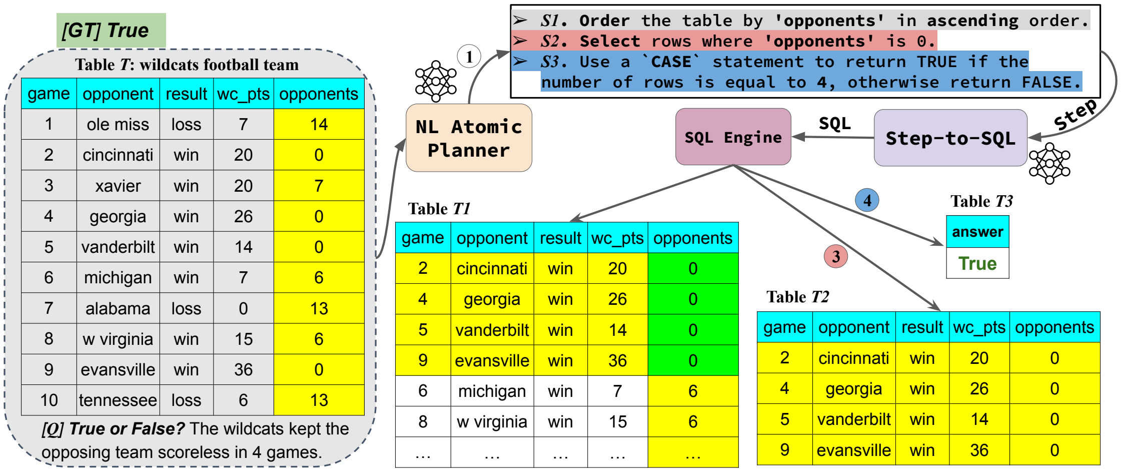
### 2.2 Program-based Table Transformations

Using programming languages such as SQL (Nahid & Rafiei, 2024b; Ye et al., 2023) or Python (Cheng et al., 2023; Chen et al., 2020) to manipulate tables is preferable for two reasons. First, these languages perform rule-based operations with explicit references to table cells, offering much greater traceability than the implicit, black-box transformations of LLMs (see contrastive examples between Fig. 1**a** & **d**). Second, programmatic transformations can handle large or complex table operations more reliably and efficiently, since they do not suffer from the context length and inconsistency issues that LLM-based methods encounter when processing entire tables (Chen, 2023; Wang et al., 2024; Nahid & Rafiei, 2024b).

`POS` builds on this line of work by leveraging program-based transformations—specifically, SQL—to solve Table QA. Notably, `POS` uses SQL *exclusively* for executing reasoning steps. To our knowledge, only two methods in the Table QA literature, LPA (Chen et al., 2020) (using Python-Pandas) and Text-to-SQL (Rajkumar et al., 2022), handle queries end-to-end through program-based operations. However, since Text-to-SQL generates a single SQL command for a query (Fig. 1**b**), it requires a highly powerful Text-to-SQL converter and often generates error-prone SQL commands (Shi et al., 2020). Similarly, LPA's one-pass program synthesis can be brittle, as generating a correct multi-step program in one go is challenging (Chen et al., 2020). In contrast, `POS` breaks the problem into multiple simpler SQL queries corresponding to atomic sub-steps expressed in natural language. This stepwise use of SQL removes the need for an advanced one-shot program generator and, as our results will show in Sec. 4, leads to higher overall accuracy and interpretability than Text-to-SQL or LPA.

---

[1] We measure user understanding by their ability to simulate and verify the model's decisions.



- ➢ *S1*. Order the table by 'opponents' in ascending order.
- ➢ *S2*. Select rows where 'opponents' is 0.
- ➢ *S3*. Use a `CASE` statement to return TRUE if the number of rows is equal to 4, otherwise return FALSE.

Table *T*: wildcats football team

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 1 | ole miss | loss | 7 | 14 |
| 2 | cincinnati | win | 20 | 0 |
| 3 | xavier | win | 20 | 7 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 6 | michigan | win | 7 | 6 |
| 7 | alabama | loss | 0 | 13 |
| 8 | w virginia | win | 15 | 6 |
| 9 | evansville | win | 36 | 0 |
| 10 | tennessee | loss | 6 | 13 |

*[Q] True or False?* The wildcats kept the opposing team scoreless in 4 games.

Table *T1*

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| … | … | … | … | … |

Table *T2*

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |

Table *T3*

| answer |
|---|
| True |

Figure 2: Illustration of Plan-of-SQLs (`POS`). ① The **Natural Language (NL) Atomic Planner** takes $(T, Q)$ as input and generates a step-by-step plan in plain language. ② **Step-to-SQL** then takes $(T, S_1)$ and converts the first step $S_1$ into an SQL query, which is executed on $T$ to produce an intermediate table $T_1$. ③ **Step-to-SQL** takes $(T_1, S_2)$ to produce and execute the next SQL query, yielding $T_2$. ④ The final **Step-to-SQL** uses $(T_2, S_3)$ to generate an SQL query that returns the final answer. We provide an interactive demo interface of `POS`.

### 2.3 Evaluating Interpretability

Interpretability is a critical aspect of AI systems, and there is a rich body of work on evaluating explanations with human users (Adebayo et al., 2020; Nguyen et al., 2021; Kim et al., 2022; Taesiri et al., 2022; Colin et al., 2022; Steyvers & Kumar, 2023; Chen et al., 2023a; Nguyen et al., 2024a;b; Zhang et al., 2025). In the context of LLM-based Table QA, TAPERA (Zhao et al., 2024) had users subjectively rate the *faithfulness* and *comprehensiveness* of explanations on a Likert scale. However, such ratings serve only as proxies for explanation quality and do not directly measure the explanations' utility to users. `POS` guarantees 100% faithfulness (the explanation steps exactly produce the answer) and comprehensiveness (no reasoning step is hidden) because its explanations explicitly represent the model's reasoning through actual executed SQL operations. Therefore, in our experiments, we prioritize direct evaluation of explanations' effectiveness—such as how well users can simulate or verify the model's behavior—over faithfulness or completeness metrics.

## 3 `POS`: Interpretable Table QA

**Problem Formulation.** In Table Question Answering (Table QA), each sample is represented as a triplet $(T, Q, A)$, where $T$ is a table, $Q$ is a query about the table, and $A$ is the answer. The task is to predict the answer $A$ given the query $Q$ and the table $T$. `POS` decomposes $Q$ into sub-queries → converts each sub-query into an SQL command → applies these commands sequentially to $T$ to arrive at $A$.

**Atomicity in Table QA Reasoning.** We define an *atomic step* as a simple, minimal table operation that can be reliably translated into an SQL statement. Specifically, each step is restricted to: (i) at most one condition (e.g., $=$), and (ii) at most one column or variable in that condition (e.g., **opponents**). Enforcing this atomicity keeps each step's translation and execution reliable and lowers the chance of errors. It also makes the model's reasoning more interpretable: because each step addresses only a small part of the query, a human can easily understand the purpose and effect of that step.

### 3.1 Planning in Natural Language

Rather than planning in a space of abstract functions for Table QA (Chen et al., 2020; Wang et al., 2024), we perform the reasoning step generation in *natural language*. Leveraging an LLM to plan in natural language takes advantage of the model's strong priors from its language training (Huang et al., 2022). It also makes the plan inherently understandable: each step is described in natural language, which is far more interpretable than, say, a sequence of opaque function names (such as **simple_query()**) whose purpose is not immediately clear (Wang et al., 2024). In Fig. 2–①, the **NL Atomic Planner** takes the input $(T, Q)$ and produces a list of atomic steps in natural language that outline how to derive the answer $A$. `POS` prompts the LLM with examples and explicit instructions to incorporate this atomic decomposition. Specifically, we use a prompt of the form:

**Generate a plan of natural-language, atomic steps**

## In-context Planning Examples

## Input Table **T**

## Query **Q**

## Planning Instructions

**Here is the plan to solve the query Q**

### 3.2 Converting Step to SQL

Once we have the plan in natural language, the next stage is to translate each described step into an executable SQL query. We harness the LLM's capability as a Text-to-SQL converter (Hong et al., 2024) to perform this conversion for each step. There are two main reasons to use a dedicated **Step-to-SQL** module (Fig. 2). First, executing the reasoning steps via SQL commands greatly improves the correctness of each step compared to letting the LLM manipulate the table implicitly in its hidden state (see our analysis in Appendix D). In particular, offloading the computation to an SQL query ensures each operation is carried out exactly and eliminates errors from the LLM's reasoning. Second, using SQL makes the sequence of transformations fully trackable: for every step, we know precisely which table entries were accessed or modified, since the SQL query specifies those conditions. This allows us to identify what information each step used.

In Fig. 2, the **Step-to-SQL** module takes the current intermediate table (initially the original table $T$) and one atomic step description as input, and outputs the corresponding SQL query. We guide the LLM to produce correct SQL with a specialized prompt template, as illustrated below:

**Convert an atomic step into SQL**

## In-context Step-to-SQL Examples

## Current Intermediate Table

## Natural-language Atomic Step

## Step-to-SQL Instructions

**Here's the SQL to execute the step**

After converting all steps, `POS` executes them sequentially using an SQL engine (we use SQLite3 (Gaffney et al., 2022) in our implementation). The intermediate result of each SQL command becomes the input for the next step, faithfully carrying out the transformations specified by the plan (see the chain Fig. 2–②→③→④). In contrast to end-to-end or "chain-of-thought" Table QA methods that rely on an LLM's latent reasoning to jump to the final answer (as in Fig. 1**a**, **c**), `POS` maintains interpretability throughout the process: the answer is obtained through another SQL operation whose effects can be understood.

### 3.3 Generating Explanations for Table QA

Despite the need for interpretability, prior work has not established a formal method to generate human-understandable explanations for LLM-based Table QA models. Most studies focus on improving accuracy and leave the model's decision-making process opaque (Wu & Feng, 2024; Kong et al., 2024; Cheng et al., 2025). To bridge this gap, we propose an approach to generate explanations for Table QA using attribution maps–the main medium for explaining AI decisions to humans in various domains, including image classification (Colin et al., 2022), text analysis (Hase & Bansal, 2020), and time series (Theissler et al., 2022).

For each step in the plan, we create an attribution map on the intermediate table that highlights the information used in that step. Specifically, as we apply a transformation, we mark the rows and columns that were selected or affected in **yellow**, and we highlight the cells that satisfy the step's condition in **green**. By doing this for every step (see Fig. 2), we obtain intermediate tables annotated with highlights indicating which data contributed to the final answer. Finally, we present the explanation as a *chain of attribution maps*: each step is shown alongside its highlighted intermediate table and a brief description in natural language. This allows a user to visually follow the process of reasoning. In practice, the explanation shows *which* cells were used *when*, making it clear *why*, for example, the final answer is **True** in the running example of Fig. 2. We also apply this visualization approach to other Table QA methods, such as CoTable (Wang et al., 2024) and DATER (Ye et al., 2023), with additional examples provided in Appendix A.

## 4 Experiments

We conduct experiments using three popular and standard Table QA benchmarks: TabFact (Chen et al., 2020), WikiTQ (Pasupat & Liang, 2015), and FeTaQA Nan et al. (2022). **TabFact** is a fact verification dataset in which each statement associated with a table is labeled TRUE or FALSE. We use the cleaned TabFact dataset from Wang et al. (2024) and evaluate Table QA methods with binary classification accuracy on the 2,024-sample test-small set. **WikiTQ** is a question-answering dataset where the goal is to answer human-written questions using an input table. Using the dataset and evaluation scripts from Ye et al. (2023), we assess model denotation accuracy (whether the predicted answer is equal to the ground-truth answer) on the 4,344-sample standard test set. **FeTaQA** is a free-form Table QA dataset where the task is to generate free-form natural language responses based on information retrieved or inferred from a table. We evaluate models on the 2,003-sample standard test set using BLEU and ROUGE.

### 4.1 Evaluating Explanations in Table QA

**Baselines.** We select Text-to-SQL (Rajkumar et al., 2022), DATER (Ye et al., 2023), and Chain-of-Table (CoTable) (Wang et al., 2024) to benchmark the interpretability of `POS`. They are chosen for their high performance, interpretability, and reproducibility (more justifications in Appendix A). Later, we compare `POS` with self-explanations (Madsen et al., 2024), where LLMs are prompted to provide explanations for their own answers.

**Evaluation measures.** To evaluate the quality of explanations, we involve both human judgments and LLM-based judgments (which we denote as LLM-as-XAI-judge). We follow the two complementary evaluation perspectives (qualitative and quantitative) proposed by Doshi-Velez & Kim (2017).

In the *qualitative* evaluation, the judge is shown the table, the question, the model's answer, and the model's explanation. The task is to rank the explanations from different methods by overall perceived preference. This preference rating is based on clearly-defined rubrics: clarity, coherence, and usefulness in understanding the model's reasoning. We refer to this as an **Preference Ranking** task, following prior works on comparative explanation evaluation (Ramaswamy et al., 2023; Yang et al., 2024; Zhang et al., 2025; Wazzan et al., 2025). For each table question, we collect a ranking of the four methods' explanations (1 = best, 4 = worst). To ensure fairness, we only consider test samples where all methods being compared got the question either correct or incorrect (so that judges are comparing explanations for answers of the same correctness). We aggregate rankings over 707 TabFact questions meeting this criterion, computing the average rank for each method (lower is better ↓).

In the *quantitative* evaluation, the judge is asked to perform tasks that objectively measure how well the explanation informs them about the model's behaviors. We use two standard tasks for evaluating explanations: *Forward Simulation* (Doshi-Velez & Kim, 2017; Hase & Bansal, 2020; Chen et al., 2022; Mills et al., 2023) and *Model Prediction Verification* (Nguyen et al., 2021; Taesiri et al., 2022; Chen et al., 2023a).

In **Forward Simulation**, the judge is given the table, question, and explanation *without the model's answer*, and must predict what answer the model's output would be. This evaluates how clearly the explanation communicates the model's decision boundary or reasoning process to users.

In **Prediction Verification**, the judge is given the table, question, model's answer, and explanation, then must decide whether the model's answer is correct or not based on the explanation. This measures how well the explanation justifies the model's prediction (e.g., can the judge catch model errors using explanations?).

We compute performance for these tasks as the percentage of samples in which the judge makes a correct decision, reported as Simulation Accuracy and Verification Accuracy, respectively.

### 4.1.1 Evaluating explanations with humans

**Motivation.** Human evaluation is considered the gold standard for evaluating AI explanations, as humans are the ultimate users who work with AI models (Doshi-Velez & Kim, 2017). We aim to study how explanations help humans in understanding then predicting model behaviors via Forward Simulation.

**Participants and Data.** We recruit 32 volunteers, all of whom are undergraduate, master, or Ph.D. students in Computer Science. In each session, users select one of four explanation methods and complete 10 samples, with an option to participate in multiple sessions. We collect 800 responses ($\approx 200$ per method).

### 4.1.2 Evaluating explanations with LLMs

**Motivation.** The use of LLMs trained to align with human preference (Ouyang et al., 2022) as judges has been gaining attention due to their strong correlation with human judgments (Dubois et al., 2024; Zheng et al., 2023; Liu et al., 2023; Mills et al., 2023; Fernández-Becerra et al., 2024; Poché et al., 2025). This makes LLM judge a promising, scalable solution for evaluating explanations, particularly in tasks like Table QA, where the information is still text-based yet structured. Thus, we are motivated to leverage LLM judges for all three tasks: Preference Ranking, Forward Simulation, and Model Prediction Verification.

**LLM judge.** Inspired by recent works showing the effectiveness of OpenAI's instruction-tuned GPT models as reliable judges (Zheng et al., 2023; Liu et al., 2023; Dubois et al., 2024), we utilize 3 OpenAI's LLMs: `gpt-4-turbo-2024-04-09`, `gpt-4o`, and `gpt-4o-mini` to evaluate Table QA explanations. We ensure the prompts encourage the model to follow the given criteria strictly (see Appendix H for the exact prompt templates and calibration procedures). We also explore the use of open-source LLMs as judges in 4.1.4.

Table 1: (a) Preference Rankings for explanation methods given by LLM judges on TabFact. Lower values indicate better rankings ↓. (b) Simulation Accuracy ↑ (%) of LLM and human judges on TabFact.

| | (a) Preference Ranking (1 → 4) ↓ | | | | (b) Simulation Accuracy (%) ↑ | | | |
|---|---|---|---|---|---|---|---|---|
| XAI method | Text-to-SQL | DATER | CoTable | `POS` | Text-to-SQL | DATER | CoTable | `POS` |
| GPT-4 | 3.33 | 3.36 | 1.98 | **1.33** | 75.15 | 80.04 | 79.99 | **84.89** |
| GPT-4o-mini | 3.95 | 2.75 | 1.75 | **1.55** | 65.67 | 73.57 | 76.53 | **81.61** |
| GPT-4o | 3.60 | 3.35 | 2.04 | **1.01** | 73.73 | 78.21 | 79.55 | **85.25** |
| Human | - | - | - | - | 83.68 | 86.50 | 84.29 | **93.00** |

Table 2: Verification Accuracy (%) ↑ of LLM judges on TabFact and WikiTQ.

| | TabFact | | | | WikiTQ | |
|---|---|---|---|---|---|---|
| XAI method | Text-to-SQL | DATER | CoTable | `POS` | DATER | `POS` |
| GPT-4 | 49.93 | 57.56 | 60.38 | **72.08** | **73.50** | 72.38 |
| GPT-4o-mini | 55.37 | 55.43 | 61.36 | **76.74** | 64.58 | **71.93** |
| GPT-4o | 55.97 | 70.95 | 67.34 | **72.85** | 73.31 | **74.45** |

### 4.1.3 Findings from the explanation evaluation

**`POS` is ranked highest in quality.** In Tab. 1(a), `POS` explanations consistently receive the best ranks from all LLM judges. Specifically, our explanations achieve average ranks of 1.33, 1.55, and 1.01 from `GPT-4`, `GPT-4o-mini`, and `GPT-4o`; respectively, substantially outperforming CoTable, DATER, and Text-to-SQL.

This shows that `POS` explanations are regarded by LLM judges as the clearest, most coherent, and most helpful for understanding the Table QA model's reasoning process (see the rubrics in Appendix H). In practice, this preference could translate into increased trust—a key factor for AI adoption in high-stakes domains (Doshi-Velez & Kim, 2017).

**`POS` is most effective for predicting model behaviors.** Tab. 1(b) shows that `POS` effectively helps human and LLM judges predict the model behaviors. Specifically, human judges achieve 93.00% with `POS` explanations, outperforming other methods such as DATER (86.50%) and CoTable (84.29%). Similarly, across all LLM judges, `POS` consistently yields the highest accuracy, with improvements ranging from 5% → 6% over the second best methods.

**`POS` is most effective for model prediction verification.** We perform this experiment on TabFact and WikiTQ, comparing the verification accuracy with different explanation methods in Tab. 2. We find that `POS` is the best method in five out of six settings. In addition, the improvements between DATER and `POS` are more pronounced in TabFact compared to WikiTQ, suggesting that the effectiveness of explanations is influenced by the nature of the task. As a more complex dataset, WikiTQ makes it inherently more difficult for the judges to verify the predictions. Please note that we do not compare `POS` with Text-to-SQL and CoTable on WikiTQ because those methods are not publicly available for this benchmark. Later in Sec. 4.1.6, we present an ablation study that further studies the factors driving `POS`'s improved interpretability.

**Qualitative rankings strongly correlate with quantitative measures.** Using Tab. 1 & Tab. 2, we perform a correlation analysis to study whether qualitative preference rankings inform quantitative measures. Since lower rankings in Tab. 1(a) indicate better explanations, we invert the rankings to align higher simulation/verification accuracy with higher preference.

Interestingly, we find *statistically significant* positive Pearson correlations between preference rankings and simulation accuracy ($r = 0.7865$, $p = 0.0024$) and vs. verification accuracy ($r = 0.7035$, $p = 0.0107$). These high correlations suggest that in Table QA, our proposed rubrics based on perceived quality (see Appendix H) can effectively identify high-utility explanations. This allows for efficient pre-selection of explanations to present to users, reducing the need for expensive and time-consuming user studies.

**Comparison with post-hoc self-explanations** Finally, we investigate whether a simple, post-hoc self-explaining approach could rival `POS` in interpretability. Tab. 10 compares `POS` with a widely used "self-explanation" method, in which the LLM first generates an answer and then retrospectively explains its reasoning (see an example in Fig. 8).

We find that self-explanation remains substantially less effective than `POS`, largely due to the *lack of faithfulness* in post-hoc explanations (Madsen et al., 2024; Chen et al., 2023b; Agarwal et al., 2024). By contrast, `POS` grounds its explanations in offline transformation steps (via SQL execution), yielding absolute faithfulness and comprehensiveness, which are especially valuable for model simulation or verification. We provide more details in Appendix B.

### 4.1.4 Evaluating explanations with open-source LLMs

So far, our evaluation relies on GPT-family models. This can restrict the generalizability of our findings in Sec. 4.1.3. To address this, we use two additional model families—**Qwen** and **Llama**—to confirm whether the observed trends in explanation quality persist across multiple LLM families.

We use TabFact as the benchmark dataset, focusing on two tasks: Forward Simulation and Model Prediction Verification. In both tasks, we test four methods—Text-to-SQL, DATER, CoTable, and `POS` —under open-source **Qwen2.5-72B-Inst** and **Llama-3.1-405B-Inst** hosted by SambaNova[2].

Table 3: Evaluating Table QA explanations (%) on TabFact with open-source models.

| **XAI Judge** | **Text-to-SQL** | **DATER** | **CoTable** | **POS** |
|---|---|---|---|---|
| *Forward Simulation Accuracy* | | | | |
| Qwen2.5-72B | 84.31% | **91.38**% | 84.18% | 90.59% |
| Llama-3.1-405B | 79.08% | 86.63% | 79.10% | **90.59**% |
| Human | 83.68% | 86.50% | 84.29% | **93.00**% |
| *Model Prediction Verification Accuracy* | | | | |
| Qwen2.5-72B | 71.57% | 75.49% | 79.25% | **80.01**% |
| Llama-3.1-405B | 75.30% | 77.57% | 77.08% | **78.49**% |

---

[2] `https://sambanova.ai/`

We report the evaluation results on open-source LLMs in Tab. 3. In Verification, `POS` consistently delivers top performance, while Text-to-SQL remains the worst across all model families. In Forward Simulation, `POS` outperforms other baselines on Llama, although DATER slightly exceeds `POS` with Qwen2.5 (91.38% vs. 90.59%). In general, we find that the interpretability provided by `POS` translates to open-source LLMs.

#### 4.1.5 LLM–Human agreement in XAI evaluation

In Tab. 4, we report the instance-level *agreement* between LLM-based and human forward simulations *on the same samples* for four explanation methods. This metric differs from accuracy: it specifically measures how often LLM judges and humans arrive at the same decisions, given identical information, even if the underlying model's final predictions might or might not be correct. We find that: First, `POS` consistently yields the highest LLM–human agreement across all tested LLM judges, reaching up to 90.59%. This suggests that when explanations are faithful and grounded—e.g., via atomic SQL steps—both LLMs and humans converge on similar decisions in evaluating explanations. Second, baseline methods exhibit lower agreement (71–83%), likely because their explanations are less informative or less faithful, leading LLM judges and humans to follow different reasoning paths. Practically, these high agreements (71–90%) suggest that LLMs may serve as effective proxies for early-phase evaluation of Table QA explanations, reducing the burden of user studies while still approximating human judgments.

Table 4: LLM–human agreement (%) on forward simulation for TabFact. Each cell indicates how often an LLM's decisions align with the human decisions on the same subset of samples (listed in Human Samples). Higher values suggest stronger agreement between LLMs and humans.

| **Method** | **GPT4o-mini** | **Qwen2.5-72B-Inst** | **Llama-3.1-405B-Inst** | **Human Samples** |
|---|---|---|---|---|
| Text-to-SQL | 71.24% | 79.08% | 75.16% | 153 |
| DATER | 81.98% | 83.14% | 76.74% | 172 |
| CoTable | 79.10% | 82.49% | 80.79% | 177 |
| `POS` | **88.30%** | **90.59%** | **89.41%** | 171 |

#### 4.1.6 Ablation study on `POS` interpretability

To better understand which components of `POS` (Sec. 3) most strongly contribute to its interpretability, we perform an ablation study on both TabFact and WikiTQ. We focus on the model prediction verification in which an XAI judge (here, `GPT-4o-mini`) is shown `POS`'s explanations and asked to verify the final answer.

We remove (i.e., ablate out) one of the following three core components in `POS` and observe the resultant drop (or change) in verification accuracy: (**i**) Atomic operations: Instead of enforcing single-condition transformations, we allow complex multi-condition steps. (**ii**) NL planning: We replace the natural-language planning with a direct prompt that asks the LLM to generate a sequence of SQL commands to solve the query. (**iii**) SQL execution: We replace the SQL-based transformations with direct black-box LLM-based transformations.

Tab. 5 shows the verification accuracy in each ablation setting. We find that removing SQL execution leads to the biggest interpretability drop (from 76.74 to 64.19% on TabFact), while removing the natural-language planning also leads to a substantial decrease. Atomic operations, though beneficial, exhibit the smallest impact when removed.

Table 5: Verification Accuracy (%) on TabFact and WikiTQ. Removing each component from `POS` reveals its impact on interpretability: SQL execution is most critical, followed by the plan, and then atomic operations.

| **Variant** | **TabFact** | **WikiTQ** |
|---|---|---|
| `POS` (Full) | 76.74% | 71.93% |
| - Atomic Operation | 76.65% (-0.09) | 71.74% (-0.19) |
| - NL Plan | 67.96% (-8.78) | 66.20% (-5.73) |
| - SQL | 64.19% (-12.55) | 66.54% (-5.39) |

We conclude that all components (atomicity, planning, and SQL) contribute meaningfully to `POS` 's improved interpretability, but SQL execution stands out as the key driver. By translating each atomic step into an explicit SQL query, `POS` naturally produces step-by-step *attribution maps* that pinpoint exactly which table cells influence the final answer. This clarity is especially valuable for verification: if a highlighted entry in the explanation is irrelevant or obviously mismatched to the question, a user can readily infer that the model's final prediction is suspect. The natural-language plan itself also matters significantly, presumably because it makes the chain-of-reasoning much easier to follow.

Table 6: Accuracy (%) for TabFact and WikiTQ using GPT-3.5 and GPT-4o-mini. "Decomposed" indicates whether queries are decomposed into sub-problems (Fig. 2–①). "Transformed by" refers to whether intermediate tables are transformed by an LLM or a program (Fig. 2–②–③). "Answered by" specifies whether the final answer is generated by an LLM or a program (Fig. 2–④). LLM-only approaches provide the final answer without table transformations. The best performance for each model and dataset is shown in **bold**.

| Method | Accuracy (%) | | Decomposed | Tables transformed by | Final answer by |
|---|---|---|---|---|---|
| | TabFact | WikiTQ | | | |
| GPT-3.5 (`gpt-3.5-turbo-16k-0613`) | | | | | |
| End-to-End QA | 70.45 | 51.84 | ✗ | - | LLM |
| Few-Shot QA | 71.54 | 52.56 | ✗ | - | LLM |
| Chain-of-Thought (Wei et al., 2022) | 65.37 | 53.48 | ✗ | - | LLM |
| Binder (Cheng et al., 2023) | 79.17 | 56.74 | ✓ | LLM + Program | Program |
| DATER (Ye et al., 2023) | 78.01 | 52.81 | ✓ | Program | LLM |
| CoTable (Wang et al., 2024) | **80.20** | **59.90** | ✓ | Program | LLM |
| Text-to-SQL (Rajkumar et al., 2022) | 64.71 | 52.90 | ✗ | Program | Program |
| LPA (Chen et al., 2020) | 68.90 | - | ✓ | Program | Program |
| `POS` (ours) | 78.31 | 54.80 | ✓ | Program | Program |
| GPT-4o-mini (`gpt-4o-mini-2024-07-18`) | | | | | |
| Binder (Cheng et al., 2023) | **84.63** | 58.86 | ✓ | LLM + Program | Program |
| DATER (Ye et al., 2023) | 80.98 | 58.83 | ✓ | Program | LLM |
| CoTable (Wang et al., 2024) | 84.24 | 55.60 | ✓ | Program | LLM |
| `POS` (ours) | 82.70 | **59.32** | ✓ | Program | Program |

### 4.2 Evaluating Table QA Performance

**Baselines** We compare `POS` with several baseline methods, categorizing them into three groups based on *how table transformation and answer generation are performed*: LLM-only, program-only, and hybrid approaches. Unless otherwise noted, we use a `temp` = 0 and `top-p` = 1 for LLM generation.

#### 4.2.1 Table QA for TabFact and WikiTQ

As shown in Tab. 6, `POS` achieves strong performance on both TabFact and WikiTQ on two different LLM backbones. When paired with GPT-3.5, `POS` achieves 78.31% accuracy on TabFact and 54.80% on WikiTQ, yielding substantial gains over LLM-only methods such as End-to-End QA, Few-Shot QA, and Chain-of-Thought. It also surpasses other program-only baselines by wide margins: for instance, `POS` outperforms Text-to-SQL by +13.6 points and LPA by +9.41 points on TabFact. Compared to hybrid approaches, which combine LLM and program-based operations, `POS` offers a compelling alternative. Although `POS` scores lower accuracy on both benchmarks than these hybrid methods, its exclusive use of program-based transformations ensures complete transparency of each reasoning step, making it much easier for users to verify and understand the underlying decision-making process.

Using GPT-4o-mini, `POS` reaches 82.70% on TabFact and **59.32%** on WikiTQ. While hybrid approaches like Binder and CoTable score slightly higher on TabFact, `POS` achieves the **best** performance on WikiTQ. This demonstrates that with a more advanced language model, `POS` remains competitive while offering great interpretability, as shown in Sec. 4.1.3.

Table 7: Accuracy of `POS` across varying table sizes. The Pearson correlation reveals negligible relationships between table size and accuracy, highlighting the robustness of `POS`.

| | TabFact | | | WikiTQ | | |
|---|---|---|---|---|---|---|
| **Size** | **Small** | **Medium** | **Large** | **Small** | **Medium** | **Large** |
| **Token Range** | 30–109 | 109–188 | 188–804 | 135–638 | 638–1307 | 1307–33675 |
| **Accuracy** | 79.1% (533/674) | 85.2% (575/675) | 81.5% (550/675) | 56.1% (713/1448) | 46.7% (574/1448) | 47.8% (558/1448) |
| **Correlation** | | −0.006 | | | −0.023 | |

#### 4.2.2 Performance vs. table size analysis

A natural concern for Table QA systems is whether the performance degrades as input tables grow larger and more complex. To answer this, we provide a quantitative evaluation of `POS` accuracy on TabFact and

WikiTQ, stratified by table size. Specifically, following the methodology in Wang et al. (2024), we sort each table by token count (our measure for table size) and split it into three bins of equal size (small, medium, and large). We use `GPT4o-mini` as the backbone LLM and report the QA accuracy in Tab. 7.

We observe that `POS` maintains robust accuracy across varying table sizes, ranging from 79.1% to 85.2%, with a negligible correlation ($r = -0.006$) between performance and table length. By contrast, WikiTQ experiences a more pronounced decline for larger tables, with accuracy dropping by approximately 10 percentage points—from 56.1% to 46.7%—and a correlation of $r = -0.023$.

We find that `POS` is more robust than existing methods—such as Binder, DATER, and CoTable—which all suffer significant accuracy drops with large tables (i.e., tables longer than 4K tokens; see Table 3 in Wang et al. (2024)). For instance, Binder and DATER experience accuracy declines of 30–50 percentage points, dropping to 6.41% and 34.62%, respectively, while CoTable degrades to 44.87% on WikiTQ. This degradation is primarily due to the challenges LLMs face when processing long contexts. In contrast, `POS`'s SQL-based executions remain resilient to variations in table length, making it well-suited for real-world scenarios where table sizes may scale to millions of tokens.

### 4.3 Efficiency Analysis

Table 8: Efficiency analysis on WikiTQ. **SC** denotes self-consistency usage, **LLM** represents the total number of LLM calls (detailed in **Breakdown**), and **DB** is the number of database queries. Notably, `POS` requires up to 25× fewer calls/queries than other methods.

| Method | SC | LLM | Breakdown | DB |
|---|---|---|---|---|
| Binder (Cheng et al., 2023) | ✓ | 50 | GenerateSQL: 50 | 50 |
| DATER (Ye et al., 2023) | ✓ | 100 | Decompose: 40;<br>Cloze: 20;<br>GenerateSQL: 20;<br>GenerateAnswer: 20 | 20 |
| CoTable (Wang et al., 2024) | ✓ | $\leq 25$ | Planning: $\leq 5$;<br>GenerateArgs: $\leq 19$;<br>GenerateAnswer: 1 | 5 |
| `POS` (ours) | ✗ | **4** | Planning: 2;<br>GenerateSQL: 2 | **2** |

Efficiency in Table QA is crucial because reducing the number of LLM calls and database queries directly lowers computational costs and improves scalability. We leverage the efficiency benchmark introduced by Wang et al. (2024), which measures the number of LLM calls required to answer a WikiTQ question. Additionally, we propose using the number of database queries (i.e., table transformations) that reflects the computational workload on the table database.

As `POS` employs highly deterministic atomic steps during planning, it eliminates the need for the costly self-consistency prompting (Wang et al., 2022) required by Binder, DATER, and CoTable. As a result, `POS` requires only four LLM calls per question—significantly fewer than CoTable (25), Binder (50), and DATER (100). Regarding database queries, `POS` is also more efficient than others with only two queries per question, compared to five of CoTable, 50 of Binder, or 20 of DATER.

### 4.4 `POS` for Free-form Table QA

`POS` processes Table QA queries end-to-end using SQL commands, performing table transformations exclusively on the input table and without accessing to external knowledge. This SQL-only pipeline restricts `POS`'s applicability to tasks requiring creativity, such as generating paragraph-like answers. To address this, we extend `POS` to the free-form Table QA task (FeTaQA) (Nan et al., 2022) by integrating an LLM call in the final step to generate free-form natural language answers, following the algorithm illustrated in Fig. 2.

We compare `POS` with End-to-End and Few-Shot QA and find that `POS` consistently outperforms both methods (see Tab. 9). This improved performance (+0.98 points in BLEU and +1.83 points in ROUGE-L compared to Few-Shot QA) is attributed to intermediate SQL executions of `POS`, which retrieve fine-grained

Table 9: Results on the FeTaQA free-form QA task (using GPT-4o-mini as the base LLM). `POS` outperforms both 0-shot end-to-end and few-shot QA.

| **Method** | **BLEU** | **ROUGE-1** | **ROUGE-L** |
|---|---|---|---|
| End-to-End QA | 18.99 | 51.92 | 46.44 |
| Few-Shot QA | 19.18 | 53.32 | 46.86 |
| `POS` | **20.16** | **54.70** | **48.69** |

and relevant information to generate final answers. In contrast, End-to-End and Few-Shot QA process the entire input table at once, making it challenging for the model to pinpoint and make use of the correct data.

Please note that we compare `POS` against these two LLM-only baselines—selected specifically to showcase the adaptability of `POS` in generating free-form natural language responses while preserving its interpretability advantages (in intermediate table transformations). As we prioritize interpretability, we have not optimized `POS` accuracy in free-form Table QA, and therefore we do not include many hybrid QA baselines in Tab. 9.

## 5 Discussion and Future Works

We address pertinent questions regarding the robustness and interpretability of `POS`; more are in Appendix D.

**How does `POS` handle real-world "messy" tables?** We have so far evaluated on benchmark tables that are semi-structured and relatively clean (as preprocessed by Wang et al. (2024) and Ye et al. (2023)). In practice, tables "in the wild" may have merged cells, nested headers, irregular formats, or other noise. `POS` in its current form does not explicitly tackle such noise. A practical extension would be to integrate a preprocessing step that normalizes and restructures messy tables (e.g. using a tool like NormTab (Nahid & Rafiei, 2024a)) before applying `POS`. This could allow `POS` to maintain high accuracy and interpretability even on more complex, non-canonical tables.

**How can `POS` detect errors in its own SQL generation (e.g. syntax errors or ambiguous column references)?** The current `POS` implementation uses a fallback: if executing a generated SQL step fails, we invoke an LLM to answer the question directly in an end-to-end manner (similar to how current Table QA works handle execution failures (Ye et al., 2023; Cheng et al., 2023; Wang et al., 2024)). To minimize such fallbacks, we propose implementing a proactive error detection and correction mechanism that validates each generated SQL query against the table schema. When an error is detected, the model automatically regenerates the query using the execution history log as context.

**Is `POS` completely interpretable?** No; no AI system is entirely interpretable. In computer vision, for example, methods such as concept bottleneck models provide human-understandable concepts (Ismail et al., 2023) for model decisions, but the reasoning underlying the network to generate the concepts remains unintelligible. Likewise, LLMs can generate chain-of-thought rationales, but the internal hidden states driving these thoughts cannot be directly understood (Wei et al., 2022; Madsen et al., 2024). `POS` bottlenecks the decision-making into human-interpretable operations—namely, natural-language steps that are translated into SQLs. They deterministically produce the answer, allowing users to follow the logical chain of table transformations. Although the underlying LLM may involve black-box internal processes (in generating natural-language steps), our focus is on making model's decision-making interpretable, aligning with prior and current interpretability research (Taesiri et al., 2022; Ismail et al., 2023; Zhao et al., 2024).

## 6 Conclusion

We introduce Plan-of-SQLs (`POS`), an interpretable, effective, and efficient approach to Table QA with large language models. `POS` decomposes a table query into simple atomic steps and executes each step using SQL, thereby ensuring that each transformation in the reasoning process is transparent. This design fills a critical gap in current LLM-based Table QA models, which often produce answers without clear explanations. Our experiments demonstrate that prioritizing interpretability does not come at the expense of performance: `POS` achieves explanation quality superior to existing methods (as confirmed by both human evaluations and LLM judges) and competitive accuracy on Table QA benchmarks. Moreover, `POS` accomplishes this with an order-of-magnitude fewer LLM calls and database queries than prior approaches, making it a more efficient and scalable solution for real-world use. We also find that human evaluators and LLM judges largely agree on their evaluations of explanations, indicating that a reliable automated evaluation is feasible. The only significant limitation we observe is rooted in the LLM's planning capability—if the LLM produces a suboptimal plan, `POS` can err, which aligns with recent observations by Zhao et al. (2024). Encouragingly, as LLMs become more powerful in planning, we expect `POS` 's performance to naturally improve while preserving its inherent interpretability. In summary, `POS` represents a step toward Table QA systems that not only deliver accurate answers but also provide human-understandable explanations.

## References

Nikhil Abhyankar, Vivek Gupta, Dan Roth, and Chandan K Reddy. H-star: Llm-driven hybrid sql-text adaptive reasoning on tables. *arXiv preprint arXiv:2407.05952*, 2024.

Julius Adebayo, Michael Muelly, Ilaria Liccardi, and Been Kim. Debugging tests for model explanations. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, pp. 700–712, 2020.

Chirag Agarwal, Sree Harsha Tanneru, and Himabindu Lakkaraju. Faithfulness vs. plausibility: On the (un) reliability of explanations from large language models. *arXiv preprint arXiv:2402.04614*, 2024.

Jayetri Bardhan, Anthony Colas, Kirk Roberts, and Daisy Zhe Wang. Drugehrqa: A question answering dataset on structured and unstructured electronic health records for medicine related queries. In *Proceedings of the Thirteenth Language Resources and Evaluation Conference*, pp. 1083–1097, 2022.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. Language models are few-shot learners, 2020.

Chacha Chen, Shi Feng, Amit Sharma, and Chenhao Tan. Machine explanations and human understanding. In *Proceedings of the 2023 ACM Conference on Fairness, Accountability, and Transparency*, pp. 1–1, 2023a.

Valerie Chen, Nari Johnson, Nicholay Topin, Gregory Plumb, and Ameet Talwalkar. Use-case-grounded simulations for explanation evaluation. *Advances in neural information processing systems*, 35:1764–1775, 2022.

Wenhu Chen. Large language models are few (1)-shot table reasoners. In *Findings of the Association for Computational Linguistics: EACL 2023*, pp. 1120–1130, 2023.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact : A large-scale dataset for table-based fact verification. In *International Conference on Learning Representations (ICLR)*, Addis Ababa, Ethiopia, April 2020.

Yanda Chen, Ruiqi Zhong, Narutatsu Ri, Chen Zhao, He He, Jacob Steinhardt, Zhou Yu, and Kathleen McKeown. Do models explain themselves? counterfactual simulatability of natural language explanations. In *Forty-first International Conference on Machine Learning*, 2023b.

Mingyue Cheng, Qingyang Mao, Qi Liu, Yitong Zhou, Yupeng Li, Jiahao Wang, Jiaying Lin, Jiawei Cao, and Enhong Chen. A survey on table mining with large language models: Challenges, advancements and prospects. *Authorea Preprints*, 2025.

Zhoujun Cheng, Tianbao Xie, Peng Shi, Chengzu Li, Rahul Nadkarni, Yushi Hu, Caiming Xiong, Dragomir Radev, Mari Ostendorf, Luke Zettlemoyer, Noah A. Smith, and Tao Yu. Binding language models in symbolic languages. *ICLR*, 2023.

Julien Colin, Thomas Fel, Rémi Cadène, and Thomas Serre. What i cannot predict, i do not understand: A human-centered evaluation framework for explainability methods. *Advances in neural information processing systems*, 35:2832–2845, 2022.

Finale Doshi-Velez and Been Kim. Towards a rigorous science of interpretable machine learning. *arXiv preprint arXiv:1702.08608*, 2017.

Yann Dubois, Percy Liang, and Tatsunori Hashimoto. Length-controlled alpacaeval: A simple debiasing of automatic evaluators. In *First Conference on Language Modeling*, 2024. URL `https://openreview.net/forum?id=CybBmzWBX0`.

Xi Fang, Weijie Xu, Fiona Anting Tan, Ziqing Hu, Jiani Zhang, Yanjun Qi, Srinivasan H. Sengamedu, and Christos Faloutsos. Large language models (LLMs) on tabular data: Prediction, generation, and understanding - a survey. *Transactions on Machine Learning Research*, 2024. ISSN 2835-8856. URL `https://openreview.net/forum?id=IZnrCGF9WI`.

Laura Fernández-Becerra, Miguel Ángel González-Santamarta, Ángel Manuel Guerrero-Higueras, Francisco Javier Rodríguez-Lera, and Vicente Matellán Olivera. Enhancing trust in autonomous agents: An architecture for accountability and explainability through blockchain and large language models. *arXiv preprint arXiv:2403.09567*, 2024.

Kevin P Gaffney, Martin Prammer, Larry Brasfield, D Richard Hipp, Dan Kennedy, and Jignesh M Patel. Sqlite: past, present, and future. *Proceedings of the VLDB Endowment*, 15(12):3535–3547, 2022.

Peter Hase and Mohit Bansal. Evaluating explainable ai: Which algorithmic explanations help users predict model behavior? In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp. 5540–5552, 2020.

Lisa Anne Hendricks, Ronghang Hu, Trevor Darrell, and Zeynep Akata. Grounding visual explanations. In *Proceedings of the European conference on computer vision (ECCV)*, pp. 264–279, 2018.

Zijin Hong, Zheng Yuan, Qinggang Zhang, Hao Chen, Junnan Dong, Feiran Huang, and Xiao Huang. Next-generation database interfaces: A survey of llm-based text-to-sql. *arXiv preprint arXiv:2406.08426*, 2024.

Wenlong Huang, Pieter Abbeel, Deepak Pathak, and Igor Mordatch. Language models as zero-shot planners: Extracting actionable knowledge for embodied agents. In *International conference on machine learning*, pp. 9118–9147. PMLR, 2022.

Aya Abdelsalam Ismail, Julius Adebayo, Hector Corrada Bravo, Stephen Ra, and Kyunghyun Cho. Concept bottleneck generative models. In *The Twelfth International Conference on Learning Representations*, 2023.

Croft Jane. Citigroup fined over 'fat finger' error that led to £1.1bn of mistaken orders | citigroup | the guardian, 2024. URL `https://www.theguardian.com/business/article/2024/may/22/citigroup-fined-over-fat-finger-error-mistaken-orders`.

Sunnie SY Kim, Nicole Meister, Vikram V Ramaswamy, Ruth Fong, and Olga Russakovsky. Hive: Evaluating the human interpretability of visual explanations. In *European Conference on Computer Vision*, pp. 280–298. Springer, 2022.

Kezhi Kong, Jiani Zhang, Zhengyuan Shen, Balasubramaniam Srinivasan, Chuan Lei, Christos Faloutsos, Huzefa Rangwala, and George Karypis. Opentab: Advancing large language models as open-domain table reasoners. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=Qa0ULgosc9`.

Yang Liu, Dan Iter, Yichong Xu, Shuohang Wang, Ruochen Xu, and Chenguang Zhu. G-eval: Nlg evaluation using gpt-4 with better human alignment. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 2511–2522, 2023.

Andrew W Lo and Jillian Ross. Generative ai from theory to practice: A case study of financial advice. *An MIT Exploration of Generative AI*, 2024.

Andreas Madsen, Sarath Chandar, and Siva Reddy. Are self-explanations from large language models faithful? In *Findings of the Association for Computational Linguistics ACL 2024*, pp. 295–337, 2024.

Edmund Mills, Shiye Su, Stuart Russell, and Scott Emmons. Almanacs: A simulatability benchmark for language model explainability. *arXiv preprint arXiv:2312.12747*, 2023.

Md Nahid and Davood Rafiei. Normtab: Improving symbolic reasoning in llms through tabular data normalization. In *Findings of the Association for Computational Linguistics: EMNLP 2024*, pp. 3569–3585, 2024a.

Md Nahid and Davood Rafiei. Tabsqlify: Enhancing reasoning capabilities of llms through table decomposition. In *Proceedings of the 2024 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, pp. 5725–5737, 2024b.

Linyong Nan, Chiachun Hsieh, Ziming Mao, Xi Victoria Lin, Neha Verma, Rui Zhang, Wojciech Kryściński, Hailey Schoelkopf, Riley Kong, Xiangru Tang, et al. Fetaqa: Free-form table question answering. *Transactions of the Association for Computational Linguistics*, 10:35–49, 2022.

Giang Nguyen, Daeyoung Kim, and Anh Nguyen. The effectiveness of feature attribution methods and its correlation with automatic evaluation scores. *Advances in Neural Information Processing Systems*, 34: 26422–26436, 2021.

Giang Nguyen, Valerie Chen, Mohammad Reza Taesiri, and Anh Nguyen. PCNN: Probable-class nearest-neighbor explanations improve fine-grained image classification accuracy for AIs and humans. *Transactions on Machine Learning Research*, 2024a. ISSN 2835-8856. URL `https://openreview.net/forum?id=OcFjqiJ98b`.

Giang Nguyen, Mohammad Reza Taesiri, Sunnie S. Y. Kim, and Anh Nguyen. Allowing humans to interactively guide machines where to look does not always improve human-ai team's classification accuracy. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 8169–8175, June 2024b.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022.

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. In *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pp. 1470–1480, 2015.

Antonin Poché, Alon Jacovi, Agustin Martin Picard, Victor Boutin, and Fanny Jourdan. Consim: Measuring concept-based explanations' effectiveness with automated simulatability. *arXiv preprint arXiv:2501.05855*, 2025.

Nitarshan Rajkumar, Raymond Li, and Dzmitry Bahdanau. Evaluating the text-to-sql capabilities of large language models. *arXiv preprint arXiv:2204.00498*, 2022.

Vikram V Ramaswamy, Sunnie SY Kim, Ruth Fong, and Olga Russakovsky. Overlooked factors in concept-based explanations: Dataset choice, concept learnability, and human capability. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 10932–10941, 2023.

Ben Richardson. Excel facts & statistics: Original research - acuity training, 3 2022. URL `https://www.acuitytraining.co.uk/news-tips/new-excel-facts-statistics-2022/`.

Tianze Shi, Chen Zhao, Jordan Boyd-Graber, Hal Daumé III, and Lillian Lee. On the potential of lexico-logical alignments for semantic parsing to sql queries. In *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1849–1864, 2020.

Stanford University; Stanford HAI. Who's at fault when ai fails in health care?, 2024. URL `https://hai.stanford.edu/news/whos-fault-when-ai-fails-health-care?utm_source=chatgpt.com`.

M Steyvers and A Kumar. Three challenges for ai-assisted decision-making. *Perspectives on Psychological Science: a Journal of the Association for Psychological Science*, pp. 17456916231181102–17456916231181102, 2023.

Mohammad Reza Taesiri, Giang Nguyen, and Anh Nguyen. Visual correspondence-based explanations improve ai robustness and human-ai team accuracy. *Advances in Neural Information Processing Systems*, 35:34287–34301, 2022.

Andreas Theissler, Francesco Spinnato, Udo Schlegel, and Riccardo Guidotti. Explainable ai for time series classification: a review, taxonomy and research directions. *Ieee Access*, 10:100700–100724, 2022.

Xuezhi Wang, Jason Wei, Dale Schuurmans, Quoc Le, Ed Chi, Sharan Narang, Aakanksha Chowdhery, and Denny Zhou. Self-consistency improves chain of thought reasoning in language models. *arXiv preprint arXiv:2203.11171*, 2022.

Zilong Wang, Hao Zhang, Chun-Liang Li, Julian Martin Eisenschlos, Vincent Perot, Zifeng Wang, Lesly Miculicich, Yasuhisa Fujii, Jingbo Shang, Chen-Yu Lee, and Tomas Pfister. Chain-of-table: Evolving tables in the reasoning chain for table understanding. In *The Twelfth International Conference on Learning Representations*, 2024. URL `https://openreview.net/forum?id=4L0xnS4GQM`.

Albatool Wazzan, Marcus Wright, Stephen MacNeil, and Richard Souvenir. Evaluating the impact of ai-generated visual explanations on decision-making for image matching. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, pp. 672–684, 2025.

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022.

Zirui Wu and Yansong Feng. Protrix: Building models for planning and reasoning over tables with sentence context. *arXiv preprint arXiv:2403.02177*, 2024.

Yuqing Yang, Boris Joukovsky, José Oramas Mogrovejo, Tinne Tuytelaars, and Nikos Deligiannis. Snippet: A framework for subjective evaluation of visual explanations applied to deepfake detection. *ACM Transactions on Multimedia Computing, Communications and Applications*, 20(8):1–29, 2024.

Yunhu Ye, Binyuan Hui, Min Yang, Binhua Li, Fei Huang, and Yongbin Li. Large language models are versatile decomposers: Decomposing evidence and questions for table-based reasoning. In *Proceedings of the 46th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pp. 174–184, 2023.

Tong Zhang, Mengao Zhang, Wei Yan Low, X Jessie Yang, and Boyang Albert Li. Conversational explanations: Discussing explainable ai with non-ai experts. In *Proceedings of the 30th International Conference on Intelligent User Interfaces*, pp. 409–424, 2025.

Yilun Zhao, Lyuhao Chen, Arman Cohan, and Chen Zhao. Tapera: Enhancing faithfulness and interpretability in long-form table qa by content planning and execution-based reasoning. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12824–12840, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in Neural Information Processing Systems*, 36:46595–46623, 2023.

# Appendix

## A Explanation Methods for Table QA

In this section, we present visual explanations for Table QA models, which help bridge the gap between model behaviors and human understanding. Each visualization provides insights into how the model leverages the input table, highlighting the key information used in its reasoning process.

For our experiments, we use TabFact (Chen et al., 2020) and WikiTQ (Pasupat & Liang, 2015), running each method on the test sets of 2,024 and 4344 samples. `POS`, CoTable, and DATER all use the same visualization format described in Sec. 3.3.

We use four different methods for explaining Table QA answers: Text-to-SQL, DATER, CoTable, and `POS` (ours). Each method offers a unique set of information as follows:

- Text-to-SQL (Rajkumar et al., 2022). This method translates the question into a single SQL command, which is then executed on the input table to produce an answer. *Explanation Generation:* We present the *generated SQL command* along with the question and the input table to show how the final answer is derived. Although the SQL itself can be clear to experts, it may require additional domain knowledge to interpret (see Fig. 4 for an example). Therefore, the we recruited Computer Science students (who have SQL expertise) for our human study.
- DATER (Ye et al., 2023). DATER solves a natural language query via extracting a relevant subtable and uses SQLs to verify partial facts (Fig. 5). *Explanation Generation:* We extract (1) the *subtable* selected by DATER from the input table, (2) the *verified facts*, and (3) *attribution maps* that highlight which table cells DATER considers relevant. Furthermore, we show step descriptions (e.g., "Select rows") from DATER's working logs, allowing users to track subtable extraction (see Fig. 5).
- CoTable (Wang et al., 2024). CoTable processes queries by planning a sequence of abstract function calls (e.g., `f_sort`, `f_select_row`, etc.), each responsible for table transformation. *Explanation Generation:* Similar to `POS`, we visualize each intermediate table along with *attribution maps*, but the transformations are represented by function names and their arguments (e.g., `f_select_row(1)`). Although this step-by-step approach can be informative, arguments are not well-justified and the final answer relies on LLM-driven "black-box" reasoning (see Fig. 6).
- `POS` (Ours). `POS` decomposes a natural language query into a sequence of *atomic* reasoning steps, where each step is explicitly translated into an executable SQL command. *Explanation Generation:* For `POS`, we present the natural language description of each step—these are functionally equivalent to the underlying SQL commands, but far more accessible and readable to users. Users can inspect each transformation through the clear and concise natural language steps, supported by attribution maps highlighting the relevant table cells for every operation (see Fig. 7).

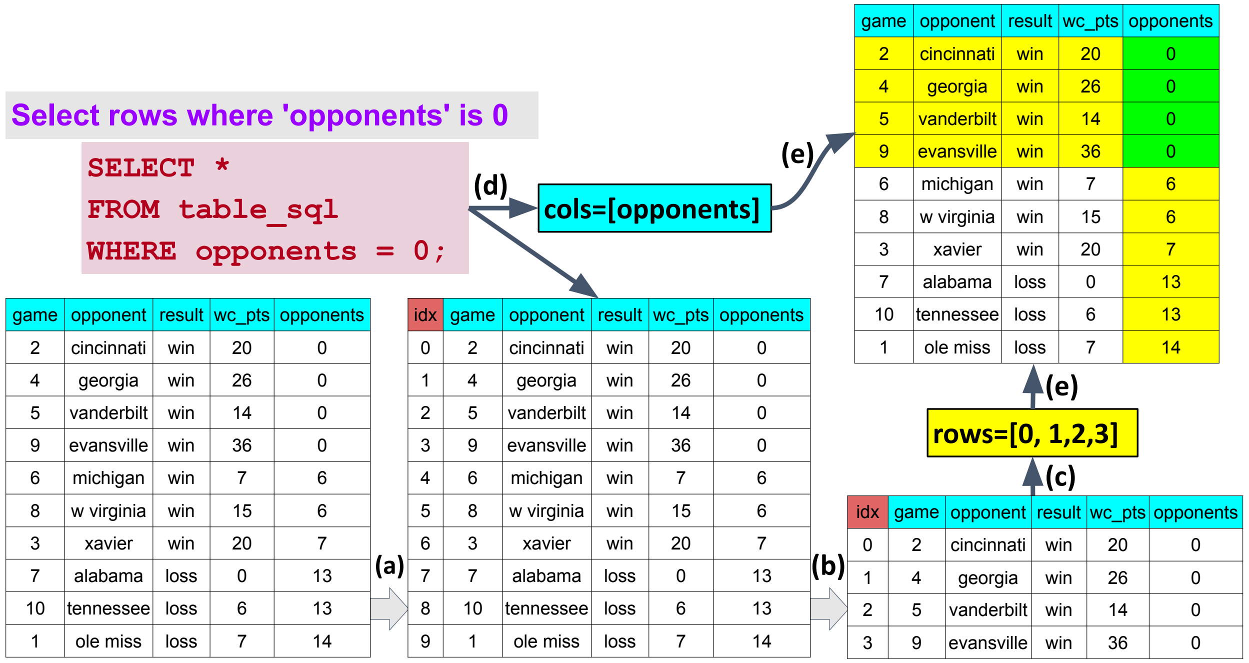
**Comparison of DATER and CoTable.** Empirically, we find no major difference in their overall "informativeness" (Sec. 4.1.3), although CoTable displays intermediate tables more explicitly. Both methods are still considered **hybrid**—they rely on partial SQLs or function calls combined with LLM-driven reasoning. For tasks demanding deeper inspection of intermediate steps (e.g., verification complex filters), CoTable's step-by-step interface may be more revealing than DATER's subtable-based approach.

**In general,** by generating method-specific explanations under a unified visualization format, we can fairly compare how each Table QA methods explain reasoning and arrives at its final answer. This shared framework allows us to examine strengths, weaknesses, and interpretability trade-offs across different approaches in earlier sections.

### A.1 Attribution Maps

During the execution of each SQL command, we perform the following steps:

- **Adding the tracking index column:** Before executing an SQL, we add a tracking index column to the current table. This column contains the original row indices from the initial table— Fig. 3**(a)**.



```sql
SELECT *
FROM table_sql
WHERE opponents = 0;
```

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| 3 | xavier | win | 20 | 7 |
| 7 | alabama | loss | 0 | 13 |
| 10 | tennessee | loss | 6 | 13 |
| 1 | ole miss | loss | 7 | 14 |

| idx | game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|---|
| 0 | 2 | cincinnati | win | 20 | 0 |
| 1 | 4 | georgia | win | 26 | 0 |
| 2 | 5 | vanderbilt | win | 14 | 0 |
| 3 | 9 | evansville | win | 36 | 0 |
| 4 | 6 | michigan | win | 7 | 6 |
| 5 | 8 | w virginia | win | 15 | 6 |
| 6 | 3 | xavier | win | 20 | 7 |
| 7 | 7 | alabama | loss | 0 | 13 |
| 8 | 10 | tennessee | loss | 6 | 13 |
| 9 | 1 | ole miss | loss | 7 | 14 |

| idx | game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|---|
| 0 | 2 | cincinnati | win | 20 | 0 |
| 1 | 4 | georgia | win | 26 | 0 |
| 2 | 5 | vanderbilt | win | 14 | 0 |
| 3 | 9 | evansville | win | 36 | 0 |

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| 3 | xavier | win | 20 | 7 |
| 7 | alabama | loss | 0 | 13 |
| 10 | tennessee | loss | 6 | 13 |
| 1 | ole miss | loss | 7 | 14 |

Figure 3: Generating attributions maps for `POS`. Column idx is added to track row attribution.

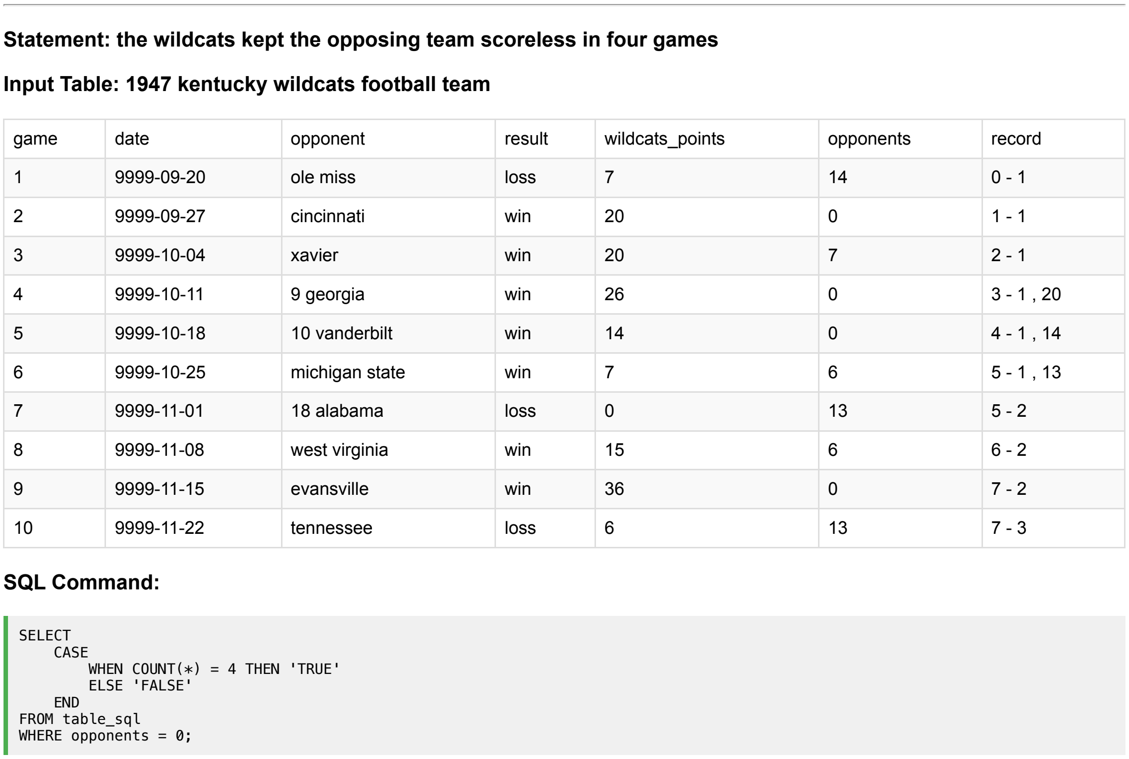
- **Executing the SQL command:** An SQL command is executed on the table with the tracking index column, producing a modified table— Fig. 3(**b**).
- **Identifying selected rows:** After execution, we use the tracking index column to identify which rows have been selected or filtered by the SQL command— Fig. 3(**c**).
- **Identifying selected columns:** We parse the SQL command to extract the columns involved in the operation— Fig. 3(**d**) (see Appendix F).
- **Visualizing an attribution map:** The prior information allow us to generate an attribution map for the intermediate tables— Fig. 3(**e**). The index column is also removed at this step.

Since both rows and columns can be attributed within an operation, `POS` offers a *distinct advantage* over previous works (Ye et al., 2023; Wang et al., 2024)—accurately attributing responsible cells for each transformation. For example, when an SQL command includes a condition that requires a cell to match a specific value or range (e.g., `WHERE opponents = 0`), we can determine which cells in the `opponents` column satisfy this condition and are thus responsible for the answer— Fig. 3(**e**).

## A.2 Explanations as Chains of Attribution Maps

At each step of the table transformation process, we visualize attribution maps on the input table for that step, highlighting the data selected or filtered in the current operation. Rows and columns that contain relevant data for the operation are yellow-highlighted, while cells that match the specific condition in this step are green-highlighted.

Using the information obtained from the plan execution and attribution maps, we combine the three components: (1) intermediate tables; (2) attribution maps; and (3) step description; to create an explanation. We present the explanation in a *chain of attribution maps*, helping users visually follow the sequence of transformations and understand how each table cell contributes to the final answer. We show representative examples of explanation methods in our study below.



**Statement: the wildcats kept the opposing team scoreless in four games**

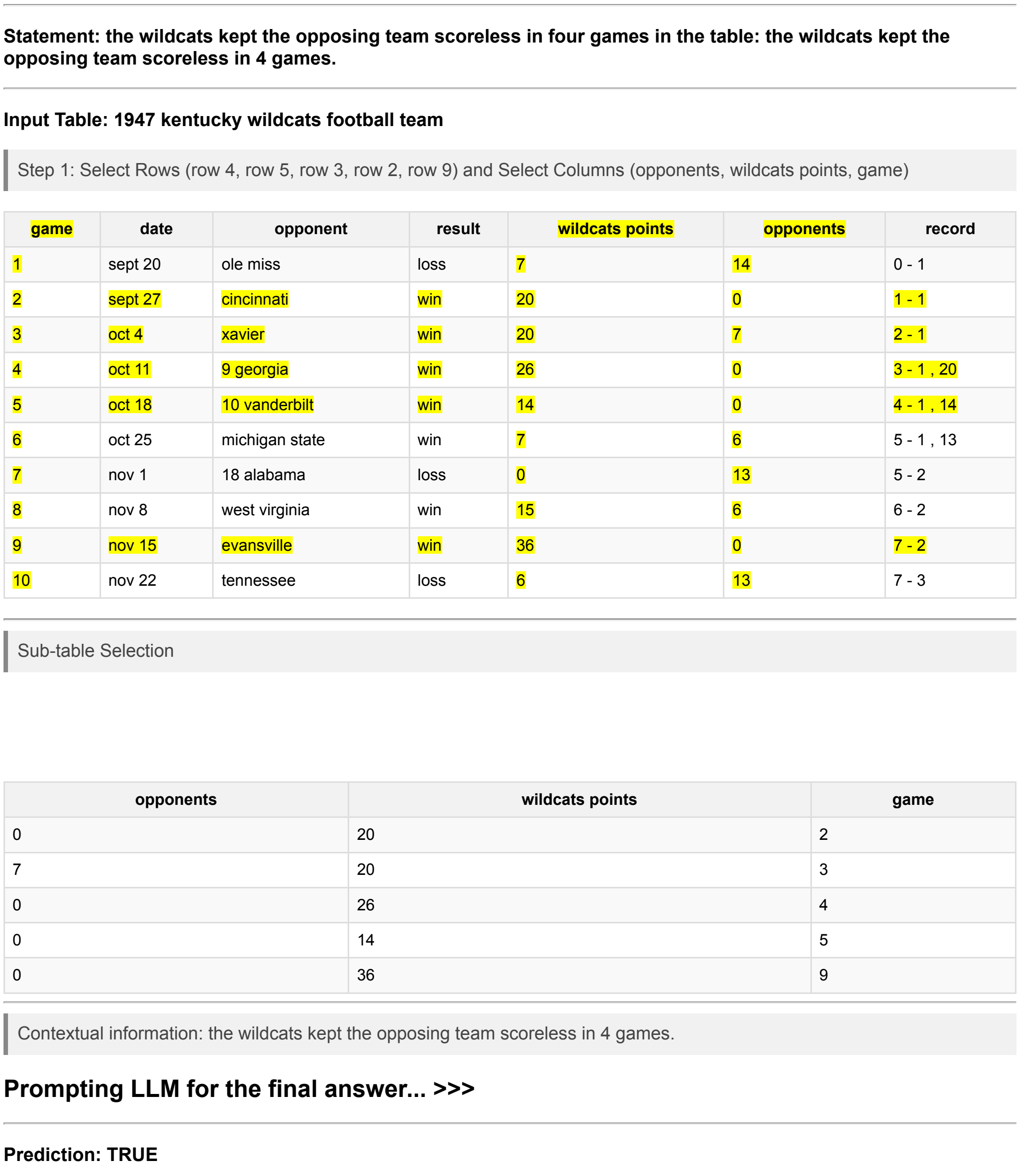
**Input Table: 1947 kentucky wildcats football team**

| game | date | opponent | result | wildcats_points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | 9999-09-20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | 9999-09-27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | 9999-10-04 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | 9999-10-11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | 9999-10-18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | 9999-10-25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | 9999-11-01 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | 9999-11-08 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | 9999-11-15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | 9999-11-22 | tennessee | loss | 6 | 13 | 7 - 3 |

**SQL Command:**

```
SELECT
    CASE
        WHEN COUNT(*) = 4 THEN 'TRUE'
        ELSE 'FALSE'
    END
FROM table_sql
WHERE opponents = 0;
```

Figure 4: **Text-to-SQL** explanations provide only the SQL command, which is intuitive for SQL users.



**Statement: the wildcats kept the opposing team scoreless in four games in the table: the wildcats kept the opposing team scoreless in 4 games.**

**Input Table: 1947 kentucky wildcats football team**

> Step 1: Select Rows (row 4, row 5, row 3, row 2, row 9) and Select Columns (opponents, wildcats points, game)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | sept 20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | oct 25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | nov 1 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | nov 8 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | nov 22 | tennessee | loss | 6 | 13 | 7 - 3 |

> Sub-table Selection

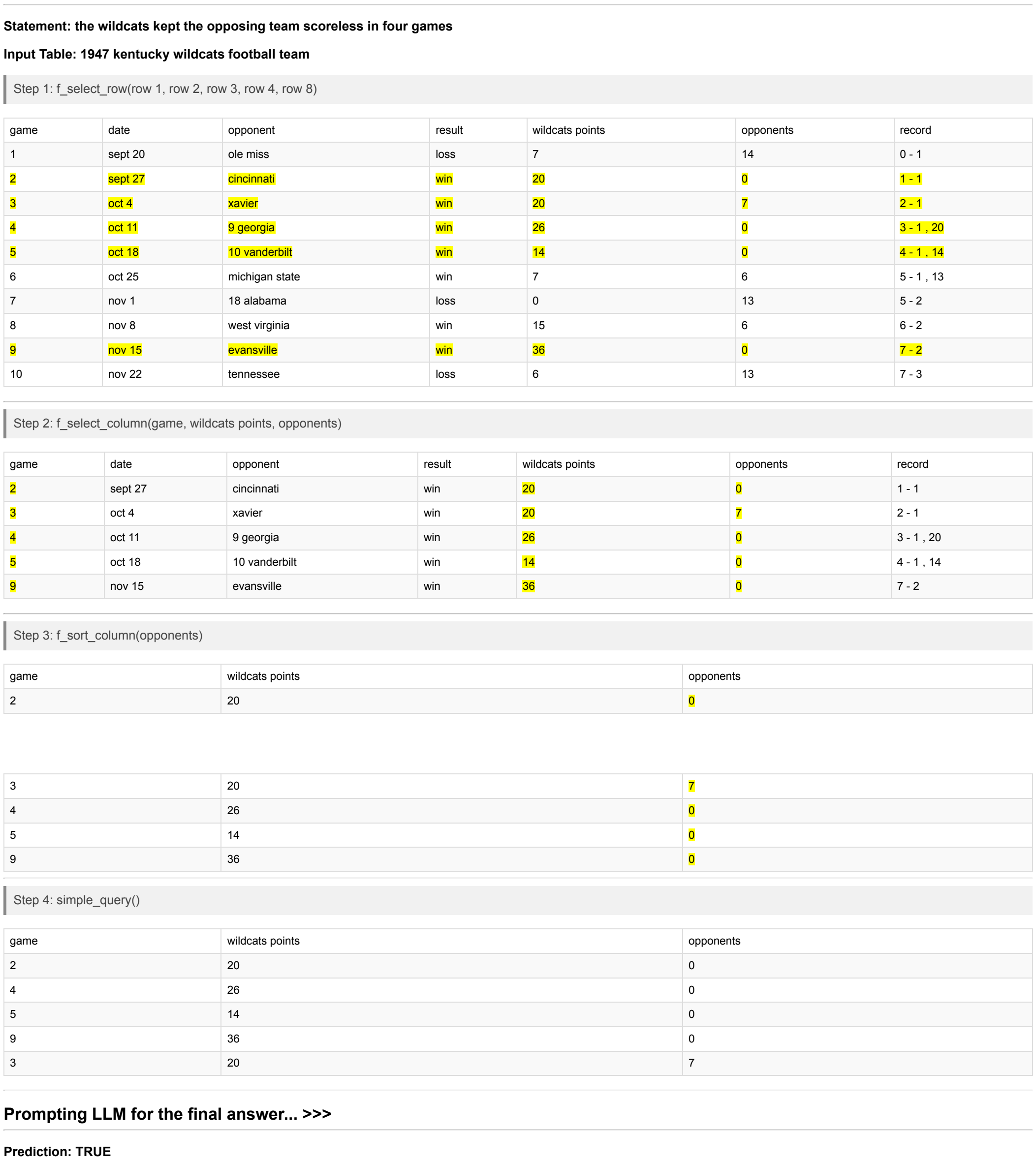
| opponents | wildcats points | game |
|---|---|---|
| 0 | 20 | 2 |
| 7 | 20 | 3 |
| 0 | 26 | 4 |
| 0 | 14 | 5 |
| 0 | 36 | 9 |

> Contextual information: the wildcats kept the opposing team scoreless in 4 games.

**Prompting LLM for the final answer... >>>**

**Prediction: TRUE**

Figure 5: **DATER** explanations contain sub-table selection, contextual information (or verified facts), and highlights that reveal which input features influence the final answer.



**Statement: the wildcats kept the opposing team scoreless in four games**

**Input Table: 1947 kentucky wildcats football team**

Step 1: f_select_row(row 1, row 2, row 3, row 4, row 8)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | sept 20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | oct 25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | nov 1 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | nov 8 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | nov 22 | tennessee | loss | 6 | 13 | 7 - 3 |

Step 2: f_select_column(game, wildcats points, opponents)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |

Step 3: f_sort_column(opponents)

| game | wildcats points | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 3 | 20 | 7 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |

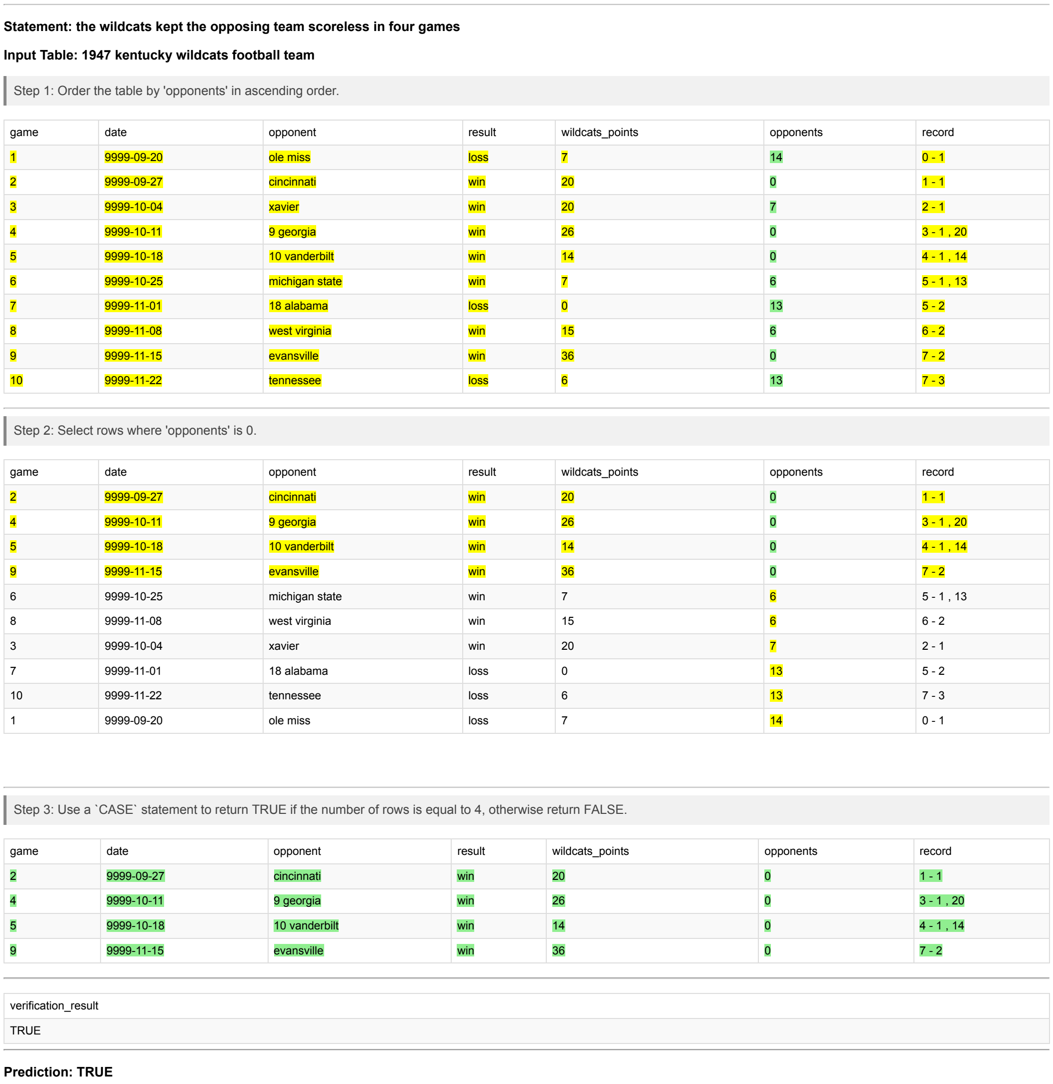
Step 4: simple_query()

| game | wildcats points | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |
| 3 | 20 | 7 |

**Prompting LLM for the final answer... >>>**

**Prediction: TRUE**

Figure 6: **CoTable** explanations present intermediate tables and highlights, showing key steps in data transformation. Additionally, the steps are presented through function names and their arguments.



**Statement: the wildcats kept the opposing team scoreless in four games**

**Input Table: 1947 kentucky wildcats football team**

Step 1: Order the table by 'opponents' in ascending order.

| game | date | opponent | result | wildcats_points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | 9999-09-20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | 9999-09-27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | 9999-10-04 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | 9999-10-11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | 9999-10-18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | 9999-10-25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | 9999-11-01 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | 9999-11-08 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | 9999-11-15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | 9999-11-22 | tennessee | loss | 6 | 13 | 7 - 3 |

Step 2: Select rows where 'opponents' is 0.

| game | date | opponent | result | wildcats_points | opponents | record |
|---|---|---|---|---|---|---|
| 2 | 9999-09-27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 4 | 9999-10-11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | 9999-10-18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 9 | 9999-11-15 | evansville | win | 36 | 0 | 7 - 2 |
| 6 | 9999-10-25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 8 | 9999-11-08 | west virginia | win | 15 | 6 | 6 - 2 |
| 3 | 9999-10-04 | xavier | win | 20 | 7 | 2 - 1 |
| 7 | 9999-11-01 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 10 | 9999-11-22 | tennessee | loss | 6 | 13 | 7 - 3 |
| 1 | 9999-09-20 | ole miss | loss | 7 | 14 | 0 - 1 |

Step 3: Use a `CASE` statement to return TRUE if the number of rows is equal to 4, otherwise return FALSE.

| game | date | opponent | result | wildcats_points | opponents | record |
|---|---|---|---|---|---|---|
| 2 | 9999-09-27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 4 | 9999-10-11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | 9999-10-18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 9 | 9999-11-15 | evansville | win | 36 | 0 | 7 - 2 |

| verification_result |
|---|
| TRUE |

**Prediction: TRUE**

Figure 7: `POS` explanations contain intermediate tables and highlights. The green-highlighted cells indicate where the information in the table matches the conditions specified in the natural language steps.

# B Self-Explanation of Table QA Models

An interesting question is whether a simple, post-hoc explanation method might achieve similar interpretability. To explore this, we adopt a widely used "self-explanation" baseline (Madsen et al., 2024; Chen et al., 2023b; Agarwal et al., 2024). Specifically, the LLM is first prompted to produce an answer and then asked to retrospectively explain how it arrived at that answer (see Fig. 8). We compare this post-hoc approach with our proposed `POS` and other XAI baselines (Text-to-SQL, DATER, and CoTable) in Table 10, focusing on three XAI benchmarks in the TabFact dataset using `GPT4o-mini`.

Table 10: Comparison of post-hoc self-explanation vs. other explanation methods. Lower is better for *Preference*; higher is better for *Forward Simulation* and *Verification*.

| **Method** | **Preference** (↓) | **Forward Sim.** (↑) | **Verification** (↑) |
|---|---|---|---|
| Self-explanation | 5.00 | 65.98% | 68.03% |
| Text-to-SQL | 3.99 | 65.67% | 55.37% |
| DATER | 2.71 | 73.57% | 55.43% |
| CoTable | 1.77 | 76.53% | 61.36% |
| `POS` (Ours) | **1.53** | **81.61%** | **76.74%** |

Our main finding is that post-hoc self-explanation outperforms Text-to-SQL (always the worst) but remains significantly less effective than our proposed `POS` method. This may be attributed to its lack of faithfulness, as extensively identified in previous studies (Chen et al., 2023b; Madsen et al., 2024; Agarwal et al., 2024), where the explanation often fails to align with the model's true reasoning process. In contrast, `POS` 's explanations are generated and *executed* at each step, inherently tying them to the actual transformations applied to the data. This design, as demonstrated in our earlier ablation and user studies, results in greater understanding and more actionable interpretability for downstream tasks like verification and forward simulation. We show an example of self-explanation below.

**Prompt:**
*You are given a Statement and an Input Table.*

*Your Task:*
*1. Verify if the Statement if True or False agaisnt the Input Table.*
*2. Explain your verification process based on the provided info.*
*3. Your answer must be TRUE or FALSE only.*

**Statement:** the wildcats kept the opposing team scoreless in four games
**Input Table Caption:** 1947 kentucky wildcats football team

| game | date | opponent | result | wildcats_points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | 9999-09-20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | 9999-09-27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | 9999-10-04 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | 9999-10-11 | 9 georgia | win | 26 | 0 | 3 - 1, 20 |
| 5 | 9999-10-18 | 10 vanderbilt | win | 14 | 0 | 4 - 1, 14 |
| 6 | 9999-10-25 | michigan state | win | 7 | 6 | 5 - 1, 13 |
| 7 | 9999-11-01 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | 9999-11-08 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | 9999-11-15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | 9999-11-22 | tennessee | loss | 6 | 13 | 7 - 3 |

**LLM Self-explanation:**
**Verification:** TRUE

*To verify the statement "the wildcats kept the opposing team scoreless in four games," we need to examine the input table, which lists the results of the Kentucky Wildcats football games. The key information here is that the Wildcats are considered to have kept the opposing team scoreless if the column "opponents_points" shows a value of "0" for that game.*
*. . .*
*We can go through the table and count how many games the Wildcats kept the opposing team scoreless (opponents_points = 0):*
*Game 1: Ole Miss – Opponents' points = 14 (not scoreless)*
*Game 2: Cincinnati – Opponents' points = 0 (scoreless)*
*Game 3: Xavier – Opponents' points = 7 (not scoreless)*
*Game 4: Georgia – Opponents' points = 0 (scoreless)*
*Game 5: Vanderbilt – Opponents' points = 0 (scoreless)*
*Game 6: Michigan State – Opponents' points = 6 (not scoreless)*
*Game 7: Alabama – Opponents' points = 13 (not scoreless)*
*Game 8: West Virginia – Opponents' points = 6 (not scoreless)*
*Game 9: Evansville – Opponents' points = 0 (scoreless)*
*Game 10: Tennessee – Opponents' points = 13 (not scoreless)*
*From the table, we see that the Wildcats kept the opposing team scoreless in four games (games 2, 4, 5, and 9).*

Figure 8: An example of a post-hoc self-explanation. The LLM first produces an answer (TRUE) and then justifies it by referencing the table entries that support the claim.

## C Baselines for Table QA on TabFact and WikiTQ

**LLM-only.** These approaches rely solely on LLMs to generate answers without explicitly performing table transformations. End-to-End QA prompts the LLM to generate answers directly from the input table and question. Similarly, Few-Shot QA (Brown et al., 2020) includes few-shot examples $(T, Q, A)$ as the context to aid the LLM. In contrast, Chain-of-Thought (Wei et al., 2022) prompts the LLM to explain its reasoning process step-by-step before delivering the final answer.

**Program-only.** Program-based approaches generate explicit programs to perform table transformation and answer the question. Latent Program Algorithm (LPA) (Chen et al., 2020) frames TabFact verification as a program synthesis task, converting input queries into sequential operations (e.g., min, max, count, filter) executed via Python-Pandas. On the other hand, Text-to-SQL (Rajkumar et al., 2022) translates a natural language query directly into a single SQL command, which is then applied to the input table to generate the answer.

**Hybrid.** Hybrid approaches combine the strengths of LLM reasoning and programs to perform Table QA and achieve state-of-the-art accuracy. DATER (Ye et al., 2023) uses an LLM to extract relevant subtables, while breaking queries into sub-queries and executing SQL commands to retrieve factual information. Binder (Cheng et al., 2023) takes a different approach by converting natural language questions into executable programs. It blends API calls with symbolic language interpreters like SQL or Python to address reasoning gaps that cannot be handled through offline methods alone. Lastly, CoTable (Wang et al., 2024) dynamically plans a sequence of predefined table operations–such as selecting rows or adding columns, allowing it to iteratively transform the table based on the intermediate information. Despite their differences, they all share a common strategy: they input the final simplified table along with the original query into an LLM to produce the final answer.

## D More Discussion and Future Works

**What is the impact of removing SQL execution from `POS`?** We investigate this impact in Sec. 4.1.6. Removing SQL execution not only diminishes interpretability, but also affects accuracy on benchmark datasets. In this ablation, the LLM is tasked with directly transforming the table rather than executing SQL commands. While this leads to a negligible decrease in Table QA accuracy on TabFact, it causes a substantial drop on WikiTQ. This discrepancy suggests that relying solely on the black-box reasoning of LLMs for table transformations can severely impact model accuracy—likely due to LLM hallucinations or errors when handling complex tables (Chen, 2023; Wang et al., 2024). Moreover, bypassing SQL removes a layer of transparency, as the table transformations are no longer traceable, a reduction in interpretability that is further quantified by Tab. 5.

**What are the most common failure modes of `POS`?** We observed that errors in **`POS`** often originate from the planning stage rather than from the Step-to-SQL. For example, the planner often omits necessary condition checks in its atomic steps. As LLMs continue to improve their planning capabilities (Huang et al., 2022), we expect **`POS`** to benefit accordingly in terms of performance while retaining its strong interpretability. Apart from planning omissions, another source of error is the strict nature of SQL matching. For example, if the table contains 'bjørn' but the statement says "bjon", an exact-match SQL filter will fail to recognize the misspelling. Please refer to Appendix K for examples and analysis of **`POS`** errors.

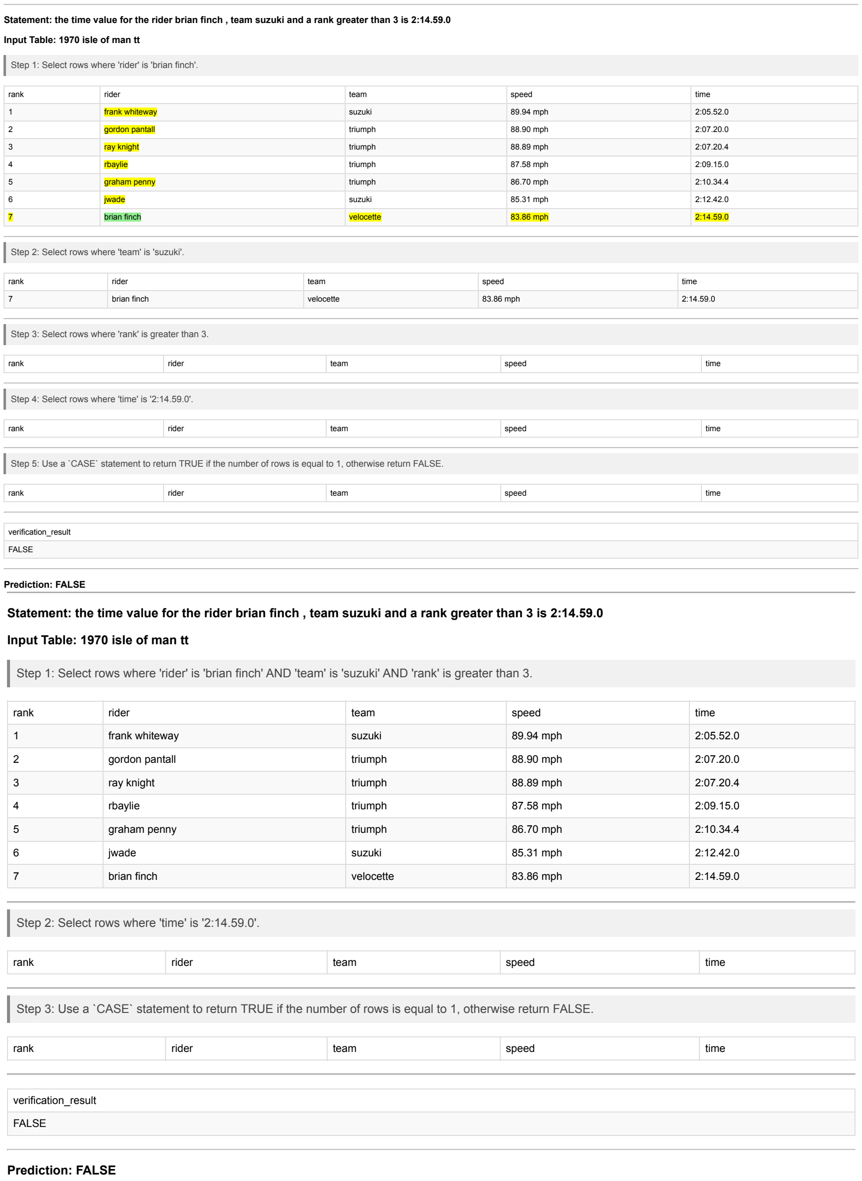
**Why `POS` explanations do not contain SQLs?** We designed **`POS`** to present natural-language steps rather than raw SQL because, in our user studies, including SQL in the explanations tended to overwhelm users. The natural language descriptions are equivalent to the SQL commands—thanks to the atomicity constraint, which guarantees minimal discrepancy between the two—yet they are far more readable and accessible. This design choice strikes a balance between interpretability and clarity, ensuring that users grasp the model's reasoning without being burdened by technical SQL syntax.

**Have we evaluated explanation effectiveness in free-form Table QA?** No; our work has not yet investigated the interpretability in free-form Table QA (e.g., FeTaQA), where the final answer is generated through a black-box operation of LLMs. A promising approach is to develop hybrid explanations that blend

**POS**-generated intermediate step explanations with groundings (Hendricks et al., 2018) for the final answer. In this hybrid format, intermediate steps generated by **POS** provide a transparent view into the intermediate reasoning process while grounding algorithms are employed to anchor the final response to the intermediate tables.

## E Qualitative Examples for POS Explanations without Atomicity

In this section, we provide qualitative examples of **POS** explanations *with* and *without* atomicity enforcement in NL Planning. Removing atomicity from the plan steps can negatively impact interpretability, as the added complexity makes it harder for users to understand the model's reasoning process.



**Statement: the time value for the rider brian finch , team suzuki and a rank greater than 3 is 2:14.59.0**

**Input Table: 1970 isle of man tt**

Step 1: Select rows where 'rider' is 'brian finch'.

| rank | rider | team | speed | time |
|---|---|---|---|---|
| 1 | frank whiteway | suzuki | 89.94 mph | 2:05.52.0 |
| 2 | gordon pantall | triumph | 88.90 mph | 2:07.20.0 |
| 3 | ray knight | triumph | 88.89 mph | 2:07.20.4 |
| 4 | rbaylie | triumph | 87.58 mph | 2:09.15.0 |
| 5 | graham penny | triumph | 86.70 mph | 2:10.34.4 |
| 6 | jwade | suzuki | 85.31 mph | 2:12.42.0 |
| 7 | brian finch | velocette | 83.86 mph | 2:14.59.0 |

Step 2: Select rows where 'team' is 'suzuki'.

| rank | rider | team | speed | time |
|---|---|---|---|---|
| 7 | brian finch | velocette | 83.86 mph | 2:14.59.0 |

Step 3: Select rows where 'rank' is greater than 3.

| rank | rider | team | speed | time |
|---|---|---|---|---|

Step 4: Select rows where 'time' is '2:14.59.0'.

| rank | rider | team | speed | time |
|---|---|---|---|---|

Step 5: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| rank | rider | team | speed | time |
|---|---|---|---|---|

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

**Statement: the time value for the rider brian finch , team suzuki and a rank greater than 3 is 2:14.59.0**

**Input Table: 1970 isle of man tt**

Step 1: Select rows where 'rider' is 'brian finch' AND 'team' is 'suzuki' AND 'rank' is greater than 3.

| rank | rider | team | speed | time |
|---|---|---|---|---|
| 1 | frank whiteway | suzuki | 89.94 mph | 2:05.52.0 |
| 2 | gordon pantall | triumph | 88.90 mph | 2:07.20.0 |
| 3 | ray knight | triumph | 88.89 mph | 2:07.20.4 |
| 4 | rbaylie | triumph | 87.58 mph | 2:09.15.0 |
| 5 | graham penny | triumph | 86.70 mph | 2:10.34.4 |
| 6 | jwade | suzuki | 85.31 mph | 2:12.42.0 |
| 7 | brian finch | velocette | 83.86 mph | 2:14.59.0 |

Step 2: Select rows where 'time' is '2:14.59.0'.

| rank | rider | team | speed | time |
|---|---|---|---|---|

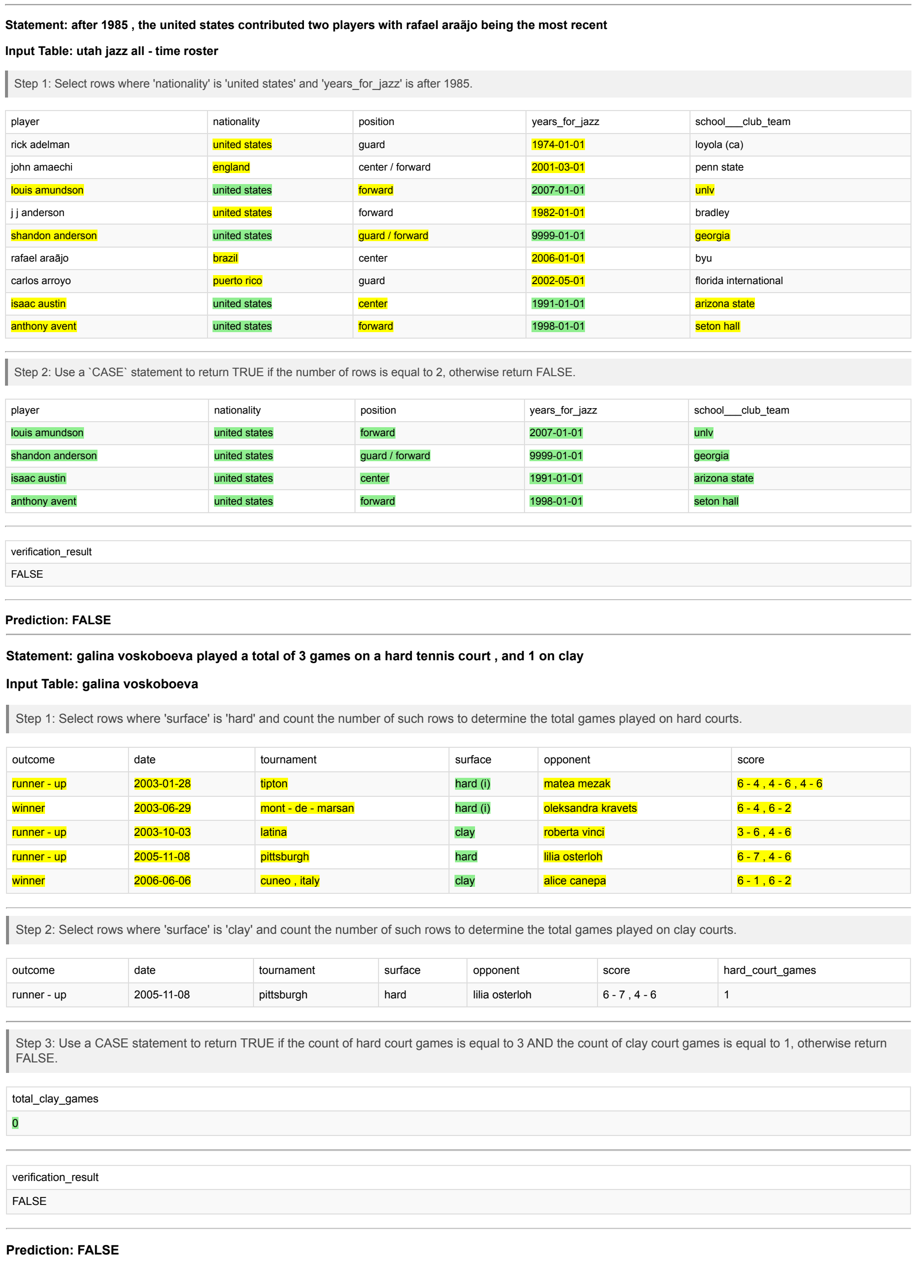
Step 3: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| rank | rider | team | speed | time |
|---|---|---|---|---|

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

Figure 9: Upper: **POS** explanation with atomicity. Lower: **POS** explanation without atomicity.



**Statement: after 1985 , the united states contributed two players with rafael araãjo being the most recent**

**Input Table: utah jazz all - time roster**

> Step 1: Select rows where 'nationality' is 'united states' and 'years_for_jazz' is after 1985.

| player | nationality | position | years_for_jazz | school___club_team |
|---|---|---|---|---|
| rick adelman | united states | guard | 1974-01-01 | loyola (ca) |
| john amaechi | england | center / forward | 2001-03-01 | penn state |
| louis amundson | united states | forward | 2007-01-01 | unlv |
| j j anderson | united states | forward | 1982-01-01 | bradley |
| shandon anderson | united states | guard / forward | 9999-01-01 | georgia |
| rafael araãjo | brazil | center | 2006-01-01 | byu |
| carlos arroyo | puerto rico | guard | 2002-05-01 | florida international |
| isaac austin | united states | center | 1991-01-01 | arizona state |
| anthony avent | united states | forward | 1998-01-01 | seton hall |

> Step 2: Use a `CASE` statement to return TRUE if the number of rows is equal to 2, otherwise return FALSE.

| player | nationality | position | years_for_jazz | school___club_team |
|---|---|---|---|---|
| louis amundson | united states | forward | 2007-01-01 | unlv |
| shandon anderson | united states | guard / forward | 9999-01-01 | georgia |
| isaac austin | united states | center | 1991-01-01 | arizona state |
| anthony avent | united states | forward | 1998-01-01 | seton hall |

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

**Statement: galina voskoboeva played a total of 3 games on a hard tennis court , and 1 on clay**

**Input Table: galina voskoboeva**

> Step 1: Select rows where 'surface' is 'hard' and count the number of such rows to determine the total games played on hard courts.

| outcome | date | tournament | surface | opponent | score |
|---|---|---|---|---|---|
| runner - up | 2003-01-28 | tipton | hard (i) | matea mezak | 6 - 4 , 4 - 6 , 4 - 6 |
| winner | 2003-06-29 | mont - de - marsan | hard (i) | oleksandra kravets | 6 - 4 , 6 - 2 |
| runner - up | 2003-10-03 | latina | clay | roberta vinci | 3 - 6 , 4 - 6 |
| runner - up | 2005-11-08 | pittsburgh | hard | lilia osterloh | 6 - 7 , 4 - 6 |
| winner | 2006-06-06 | cuneo , italy | clay | alice canepa | 6 - 1 , 6 - 2 |

> Step 2: Select rows where 'surface' is 'clay' and count the number of such rows to determine the total games played on clay courts.

| outcome | date | tournament | surface | opponent | score | hard_court_games |
|---|---|---|---|---|---|---|
| runner - up | 2005-11-08 | pittsburgh | hard | lilia osterloh | 6 - 7 , 4 - 6 | 1 |

> Step 3: Use a CASE statement to return TRUE if the count of hard court games is equal to 3 AND the count of clay court games is equal to 1, otherwise return FALSE.

| total_clay_games |
|---|
| 0 |

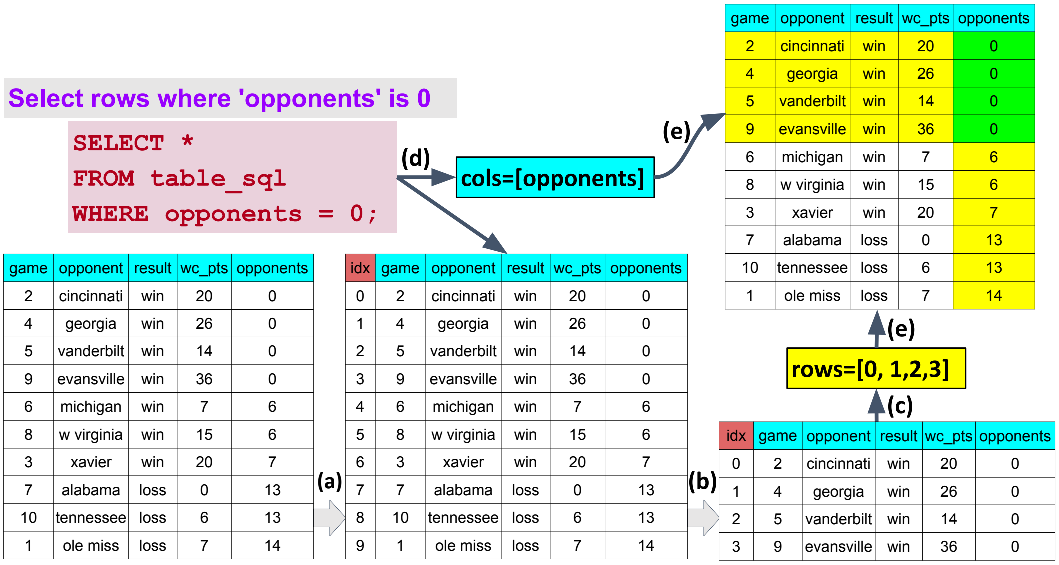
| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

Figure 10: Two `POS` explanations without atomicity. The steps are compound and the attribution maps are non-trivial to comprehend.

# F Extracting Columns from SQL Commands

In this section, we detail the algorithm to analyze SQL queries and identify the columns used within them.



Select rows where 'opponents' is 0

```
SELECT *
FROM table_sql
WHERE opponents = 0;
```

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| 3 | xavier | win | 20 | 7 |
| 7 | alabama | loss | 0 | 13 |
| 10 | tennessee | loss | 6 | 13 |
| 1 | ole miss | loss | 7 | 14 |

| idx | game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|---|
| 0 | 2 | cincinnati | win | 20 | 0 |
| 1 | 4 | georgia | win | 26 | 0 |
| 2 | 5 | vanderbilt | win | 14 | 0 |
| 3 | 9 | evansville | win | 36 | 0 |
| 4 | 6 | michigan | win | 7 | 6 |
| 5 | 8 | w virginia | win | 15 | 6 |
| 6 | 3 | xavier | win | 20 | 7 |
| 7 | 7 | alabama | loss | 0 | 13 |
| 8 | 10 | tennessee | loss | 6 | 13 |
| 9 | 1 | ole miss | loss | 7 | 14 |

| idx | game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|---|
| 0 | 2 | cincinnati | win | 20 | 0 |
| 1 | 4 | georgia | win | 26 | 0 |
| 2 | 5 | vanderbilt | win | 14 | 0 |
| 3 | 9 | evansville | win | 36 | 0 |

| game | opponent | result | wc_pts | opponents |
|---|---|---|---|---|
| 2 | cincinnati | win | 20 | 0 |
| 4 | georgia | win | 26 | 0 |
| 5 | vanderbilt | win | 14 | 0 |
| 9 | evansville | win | 36 | 0 |
| 6 | michigan | win | 7 | 6 |
| 8 | w virginia | win | 15 | 6 |
| 3 | xavier | win | 20 | 7 |
| 7 | alabama | loss | 0 | 13 |
| 10 | tennessee | loss | 6 | 13 |
| 1 | ole miss | loss | 7 | 14 |

Figure 11: Data-attribution tracking algorithm for `POS`.

## F.1 Algorithm Overview

The algorithm follows these main steps:

1. **Preprocessing:** Remove comments and normalize whitespace in the SQL query.
2. **Column Extraction:** Parse different clauses of the SQL query to identify column names:
    - `SELECT` clause: Extract both regular columns and those used in functions.
    - `WHERE` clause: Identify columns used in conditions.
    - `ORDER BY` clause: Extract columns used for sorting.
3. **Filtering:** Compare extracted columns against a list of original columns to ensure validity.

## F.2 Implementation Details

The algorithm is implemented using regular expressions to parse the SQL query. Key implementation details include:

- Use of `re.sub()` for comment removal and whitespace normalization.
- Application of `re.search()` and `re.findall()` for extracting column names from different parts of the query.
- Special treatment for columns used within functions in the `SELECT`, `WHERE`, `ORDER BY` clauses.

## F.3 An example of data-attribution tracking for Table QA

Here, we use the table transformation in Fig. 2–③ as an example to illustrate our data-attribution tracking algorithm (Fig. 11):

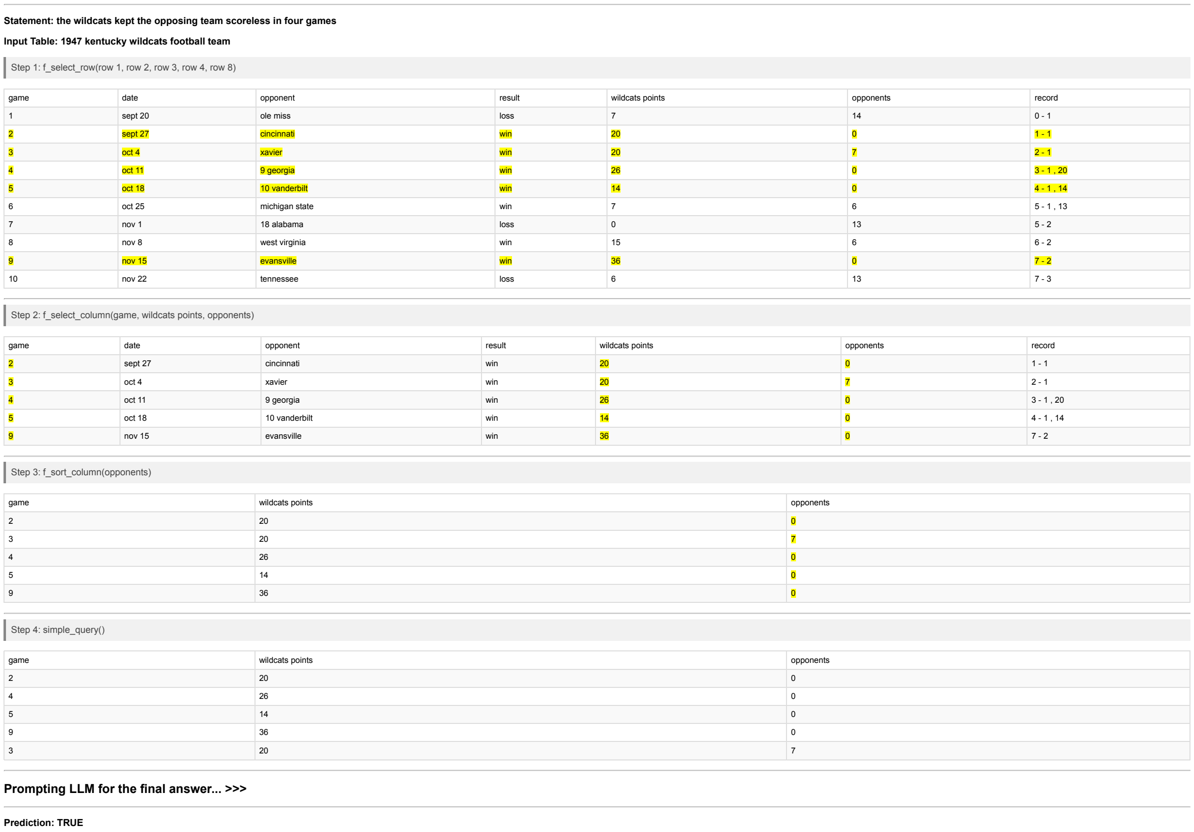
- **(a)** Adding the Tracking Index Column

- **(b)** Executing the SQL Command
- **(c)** Identifying Selected Rows
- **(d)** Parsing SQL Commands to Identify Selected Columns
- **(e)** Mapping to Original Indices

## G Hallucinations in Sub-table Selection

Methods like CoTable and DATER aim to answer questions by performing complex table transformations—specifically, selecting sub-tables from the input table based on reasoning steps. However, these methods are prone to errors regarding which table entries to select, leading to irrational or irrelevant information being considered in the final answer.

As illustrated in Fig. 12, although Chain-of-Table correctly answers the question ***Q****: True or False? In four different baseball games, the final score was 9-2*, it irrationally selects unrelated information (game 3) from the input table. Similarly, DATER, shown in Fig. 13, selects rows 2, 3, 4, 5, and 9 to answer the same question. However, the inclusion of row 3 is illogical and does not contribute to a valid answer.



**Statement: the wildcats kept the opposing team scoreless in four games**

**Input Table: 1947 kentucky wildcats football team**

Step 1: f_select_row(row 1, row 2, row 3, row 4, row 8)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | sept 20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | oct 25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | nov 1 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | nov 8 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | nov 22 | tennessee | loss | 6 | 13 | 7 - 3 |

Step 2: f_select_column(game, wildcats points, opponents)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |

Step 3: f_sort_column(opponents)

| game | wildcats points | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 3 | 20 | 7 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |

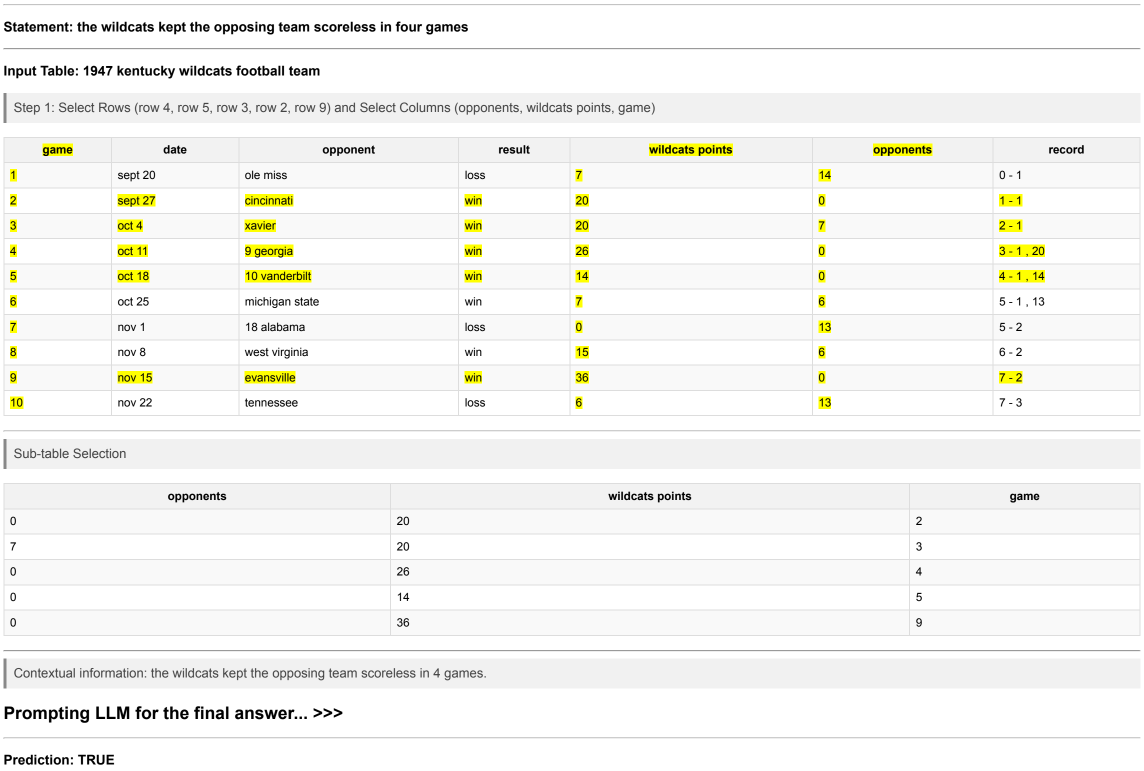
Step 4: simple_query()

| game | wildcats points | opponents |
|---|---|---|
| 2 | 20 | 0 |
| 4 | 26 | 0 |
| 5 | 14 | 0 |
| 9 | 36 | 0 |
| 3 | 20 | 7 |

**Prompting LLM for the final answer... >>>**

**Prediction: TRUE**

Figure 12: Although CoTable correctly answers the question ***Q****: True or False? In four different baseball games, the final score was 9-2*, it irrationally selects unrelated information (game 3) from the input table.



**Statement: the wildcats kept the opposing team scoreless in four games**

**Input Table: 1947 kentucky wildcats football team**

Step 1: Select Rows (row 4, row 5, row 3, row 2, row 9) and Select Columns (opponents, wildcats points, game)

| game | date | opponent | result | wildcats points | opponents | record |
|---|---|---|---|---|---|---|
| 1 | sept 20 | ole miss | loss | 7 | 14 | 0 - 1 |
| 2 | sept 27 | cincinnati | win | 20 | 0 | 1 - 1 |
| 3 | oct 4 | xavier | win | 20 | 7 | 2 - 1 |
| 4 | oct 11 | 9 georgia | win | 26 | 0 | 3 - 1 , 20 |
| 5 | oct 18 | 10 vanderbilt | win | 14 | 0 | 4 - 1 , 14 |
| 6 | oct 25 | michigan state | win | 7 | 6 | 5 - 1 , 13 |
| 7 | nov 1 | 18 alabama | loss | 0 | 13 | 5 - 2 |
| 8 | nov 8 | west virginia | win | 15 | 6 | 6 - 2 |
| 9 | nov 15 | evansville | win | 36 | 0 | 7 - 2 |
| 10 | nov 22 | tennessee | loss | 6 | 13 | 7 - 3 |

Sub-table Selection

| opponents | wildcats points | game |
|---|---|---|
| 0 | 20 | 2 |
| 7 | 20 | 3 |
| 0 | 26 | 4 |
| 0 | 14 | 5 |
| 0 | 36 | 9 |

Contextual information: the wildcats kept the opposing team scoreless in 4 games.

**Prompting LLM for the final answer... >>>**

**Prediction: TRUE**

Figure 13: DATER selects rows 2, 3, 4, 5, and 9 to answer the question. However, the inclusion of row 3 is illogical and does not contribute to a valid answer.

## H Details for LLM-as-a-Judge Experiments

In this section, we describe how we prompt LLMs to work as judges for evaluating explanation methods.

### H.1 Prompt to LLM Judges in Prediction Verification

**Prompt for LLM-as-a-Judge in Prediction Verification**

prompt = f"""

The Table Fact Verification (TabFact) model is working on verifying if a given Statement is TRUE or FALSE based on a given input Table.

You are given an HTML file containing a Statement, Input Table, Prediction, and an Explanation clarifying the Prediction.

Your task is to carefully analyze the Explanation and determine whether the Prediction is correct or not.

Explanation Method: [**method**]

[**method_specific_info**]

HTML content: [**html_content**]

Answer with ’option1’ or ’option2’ only.

You MUST ignore the order of the options and answer based on the correctness of the Prediction!

"""

### H.2 Prompt to LLM Judges in Forward Simulation

**Prompt for LLM-as-a-Judge in Forward Simulation**

prompt = f"""

Given an input statement, an Artificial Intelligence (AI) model will output either TRUE or FALSE. Your job in this Simulation task is to use the AI’s explanation to guess the machine response. Specifically, please choose which response (TRUE/FALSE) model would output regardless of whether you think that response is correct or not.

Explanation: [**text_content**]

Based on this explanation, guess what the model will predict on the Statement based on the provided explanation. Answer with only ‘TRUE’ or ‘FALSE’:

"""

### H.3 Prompt to LLM Judges in Preference Ranking

It is well known that LLM-as-a-Judge exhibits a strong bias toward the position of the options presented to it (Dubois et al., 2024). To eliminate this bias in our prompt, we shuffle the order of the four methods four times and compute the average ranking.

**Prompt for LLM-as-a-Judge in Preference Ranking**

```
    prompts = []

    num_methods = len(methods)
```

# Create a dictionary mapping methods to their descriptions

method_descriptions = {

"DATER": """DATER is a method that focuses on selecting relevant information from the input table and providing contextual information to support the statement verification process. The explanation contains:

1. Sub-table Selection: DATER selects a sub-table from the original input Table that is relevant to the Statement.

2. Contextual Information: DATER provides contextual information that is fact-checked against the Table.""",

"COT": """COT is a method that breaks down the question-answering process into a series of intermediate tables. Each step in the chain represents a specific operation on the table, leading to the final answer. The explanation contains:

1. Step Descriptions: Each step is accompanied by a function with arguments, providing context for the transformation.

2. Intermediate Tables: We display the intermediate tables resulting from each function, showing the state of the data at each step.

3. Row and Column Highlighting: Rows and Columns used in the current step are highlighted with background-color:yellow.""",

"Text2SQL": """Text2SQL is a method that translates the natural language query into a single SQL query. The SQL query itself serves as the explanation for how the system arrives at its answer. The explanation contains: The generated SQL command that will be directly applied onto the table to generate the final answer.""",

"POS": """POS is a Table QA method that breaks down the question-answering process into a series of natural-language steps. Each step represents a specific operation on the table, leading to the final answer. The explanation contains:

1. Step Descriptions: Each step is accompanied by a natural-language description of the atomic step performed, providing context for the transformation.

2. Intermediate Tables: We display the intermediate tables resulting from each step, showing the state of the data at each step.

3. Attribution Maps: We highlight the the rows, columns, and cells involved in each table transformation over intermediate tables. Row and Column Highlighting: Rows and Columns used in the current step are highlighted with background-color:yellow. Cell Highlighting: Cells that directly match the conditions in the current step are highlighted with background-color:90EE90.""" }

```
    for i in range(num_methods):
        shuffled_methods = methods[i:] + methods[:i]
```

prompt = f""" You are given explanations from four different methods for the same table fact verification task. Please rank these explanations based on their clarity, coherence, and helpfulness in understanding the model's reasoning.

Clarity Definition: How easy is the explanation to understand? Is the language clear and straightforward?

Coherence Definition: Does the explanation logically flow and make sense as a whole? Are the ideas well-connected?

Helpfulness in Understanding the Model's Reasoning Definition: How effectively does the explanation help you understand why the model made its decision? Does it reveal the reasoning process?

Provide the ranking from best to worst.

Explanations:

"""

# I Prompt Engineering

## I.1 Prompt for Atomic Planning

### I.1.1 Decomposition of Query $Q$

The decomposition process breaks down the complex query $Q$ into a sequence of atomic steps. This is achieved through a carefully crafted prompt provided to the LLM. The prompt includes:

- **Instructional Guidelines:** We instruct the LLM to “Develop a step-by-step plan to answer the question given the input table”.
- **Emphasis on Atomicity:** The LLM is instructed that “Each step in your plan should be very atomic and straightforward, ensuring they can be easily executed or converted into SQL”.
- **In-context Examples:** We provide example inputs $(T, Q)$ along with their corresponding plans to serve as in-context examples for planning (see Appendix J).

### I.1.2 Sequencing of Steps

Correct sequencing is crucial because each step depends on the output of the previous one. We ensure proper sequencing by:

- **Explicit Instructions:** The LLM is instructed that “The order of steps is crucial! You must ensure the orders support the correct information retrieval and verification!”.
- **Dependencies:** Clarifying that “The next step will be executed on the output table of the previous step. The first step will be executed on the given Table”.
- **Handling Comparatives and Superlatives:** Instructing the LLM on how to handle statements involving terms like ‘highest’, ‘lowest’, etc., by ordering the table before selecting rows.

**Prompt for atomic planning**

[**In-context examples**]

**### Here come to your task!**

**Table caption:** {caption}

`/*` {table2string(table_info["table_text"])} `*/` # Convert Table into markdown format

**This Table has {num_rows} rows.**

**Statement:** {sample["statement"]}

Let’s develop a step-by-step plan to verify if the given **Statement** is **TRUE** or **FALSE** on the given **Table**!

You **MUST** think carefully analyze the **Statement** and comprehend it before writing the plan!

**Plan Steps:** Each step in your plan should be very atomic and straightforward, ensuring they can be easily executed or converted into SQL.

You **MUST** make sure all conditions (except those mentioned in the table caption) are checked properly in the steps.

**Step order:** The order of steps is crucial! You must ensure the orders support the correct information retrieval and verification!

The next step will be executed on the output table of the previous step. The **first step** will be executed on the given **Table**.

For comparative or superlative **Statement** involving "highest," "lowest," "earliest," "latest," "better," "faster," "earlier," etc., you should order the table accordingly before selecting rows. This ensures that the desired comparative or superlative data is correctly retrieved.

**Plan:**

### I.1.3 The Importance of the Step Order

In this example, step 1 is crucial. If the table is not ordered by ‘rank’ first, selecting row number 1 (step 2) or filtering by ‘athlete’ (step 3) will return the wrong result. Only by ensuring that the table is correctly ordered beforehand can we reliably select the top-ranked athlete. Thus, the sequence of steps must be followed precisely to avoid logical errors.

**A plan where the step order determines the correctness**

**Table:** Olympic 2018; Table Tennis

```
/*
col : rank| athlete                | time
row 1 : 1 | manjeet kaur (ind)     | 52.17
row 2 : 2 | olga tereshkova (kaz)  | 51.86
row 3 : 3 | pinki pramanik (ind)   | 53.06
*/
```

**Statement:** manjeet had the highest rank in the competition.

**Plan:**

1. Order the table by ‘rank’ in ascending order.
2. Select row number 1.
3. Select rows where ‘athlete’ is ‘manjeet’ using the `LIKE` function.
4. Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

## I.2 Prompt for Step-to-SQL

**Prompt for Step-to-SQL**

[**In-context examples**]

Given this table:

`/*` {table2string(intermediate_table)} `*/`

Data types of columns:

- {col_1}: {dtype_str_1}
- {col_2}: {dtype_str_2}
- …

Write an SQL command that: {natural_language_step}

The original table has {num_rows} rows.

**Constraints for your SQL:**

1. If using SELECT COUNT(*), SUM, MAX, AVG, you MUST use AS to name the new column. If adding new columns, they should be different than columns {existing_cols}.
2. Your SQL command **MUST** be compatible and executable by Python `sqlite3` and `pandas`.
3. If using `FROM`, the table to be selected **MUST** be {table_name}.

# J In-context Examples

## J.1 In-context Examples for Atomic Planning

**In-context examples for atomic planning**

**TabFact**

**Table:** 2005 tournament results

```
/*
col   : id | name    | hometown      | score
row 1 : 1 | alice   | new york      | 85
row 2 : 2 | bob     | los angeles   | 90
row 3 : 3 | charlie | chicago       | 75
row 4 : 4 | dave    | new york      | 88
row 5 : 5 | eve     | los angeles   | 92
*/
```

**Statement:** in 2005 tournament, bob and charlie are both from chicago.

**Plan:** # Natural-language step

1. Select rows where the ‘name’ is ‘bob’ or ‘charlie’.
2. Select rows where ‘hometown’ is ‘chicago’.
3. Use a `CASE` statement to return TRUE if the number of rows is equal to 2, otherwise return FALSE.

**WikiTQ**

**Table:** 2005 tournament results

```
/*
col   : id | name    | hometown      | score
row 1 : 1 | alice   | new york      | 85
row 2 : 2 | bob     | los angeles   | 90
row 3 : 3 | charlie | chicago       | 75
*/
```

**Question:** which players are from chicago?

**Plan:** # Natural-language step

1. Select rows where the ‘hometown’ is ‘chicago’.
2. Select the ‘name’ column.

### J.2 In-context Examples for Step-to-SQL

**In-context examples for Step-to-SQL**

Given this table:

```
/*
col : id  | name    | department | salary | years
row 1 : 1 | alice   | it         | 95000  | 3
row 2 : 2 | bob     | finance    | 105000 | 5
row 3 : 3 | charlie | marketing  | 88000  | 2
*/
```

Write an SQL command that: Select rows where the ‘salary’ is greater than 100000.

SQL is:

```sql
SELECT *
FROM table_sql
WHERE salary > 100000;
-- Select rows where the ‘salary’ is greater than 100000.
```

## K Error Analysis of `POS` and Improved Planning Algorithm

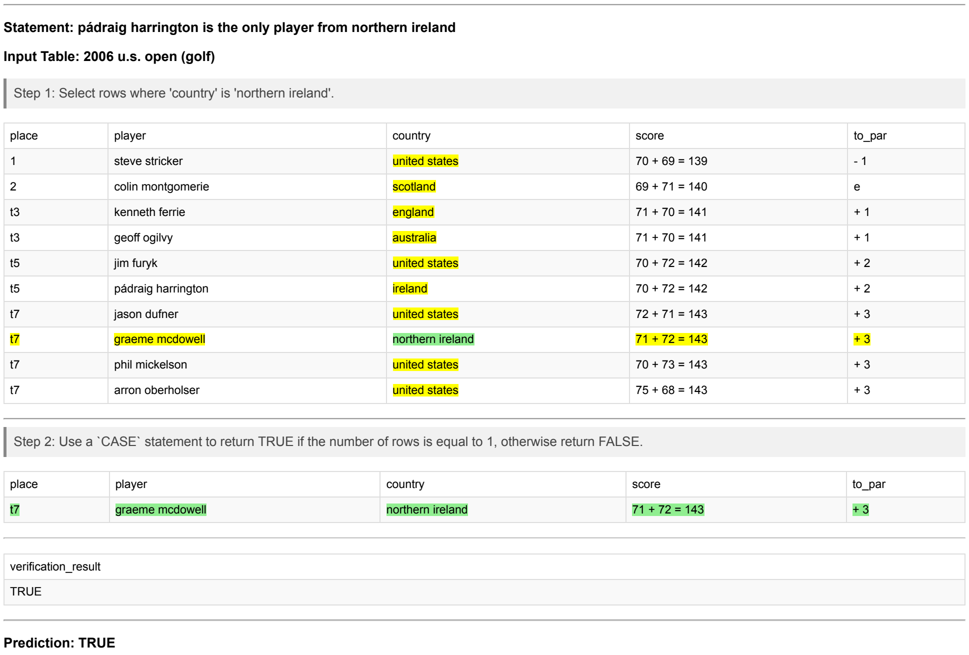
We notice that many errors in **POS** are due to the planning stage rather than the Step-to-SQL process. In particular, the Planner misses condition checks (see Fig. 14, Fig. 16, Fig. 15, Fig. 17, Fig. 18) in atomic steps. Another interesting (and inherently unavoidable) error is due to the exact-matching nature of SQL can also be found in Fig. 19.

Based on this observation, we implement an improved planning algorithm in which *only one step is generated at a time*, rather than generating all steps upfront, as shown in Fig. 1. This approach encourages the LLM to think one step at a time and reduces the complexity of the planning task. The input of NL Planner is the previous steps and the current intermediate table.

As shown in Tab. 11, planning one step at a time leads to substantial improvements in accuracy for both TabFact and WikiTQ with `GPT-4o-mini`. Please note that this one-step planning should not change the interpretability of **POS**. For the lack of proper baselines, we do not put this result into the main text, but the future works should consider using one-step planning over one-time planning.

Table 11: Table QA accuracy (%) in TabFact and WikiTQ using planning one step at a time (one-step planning) with `GPT-4o-mini`.

| **Method** | **TabFact** | **WikiTQ** |
|---|---|---|
| End-to-end QA | 71.17 | 49.24 |
| POS one-**time** planning | 77.22 | 48.90 |
| POS one-**step** planning | 83.45 | 59.32 |



**Statement: pádraig harrington is the only player from northern ireland**

**Input Table: 2006 u.s. open (golf)**

Step 1: Select rows where 'country' is 'northern ireland'.

| place | player | country | score | to_par |
|---|---|---|---|---|
| 1 | steve stricker | united states | 70 + 69 = 139 | - 1 |
| 2 | colin montgomerie | scotland | 69 + 71 = 140 | e |
| t3 | kenneth ferrie | england | 71 + 70 = 141 | + 1 |
| t3 | geoff ogilvy | australia | 71 + 70 = 141 | + 1 |
| t5 | jim furyk | united states | 70 + 72 = 142 | + 2 |
| t5 | pádraig harrington | ireland | 70 + 72 = 142 | + 2 |
| t7 | jason dufner | united states | 72 + 71 = 143 | + 3 |
| t7 | graeme mcdowell | northern ireland | 71 + 72 = 143 | + 3 |
| t7 | phil mickelson | united states | 70 + 73 = 143 | + 3 |
| t7 | arron oberholser | united states | 75 + 68 = 143 | + 3 |

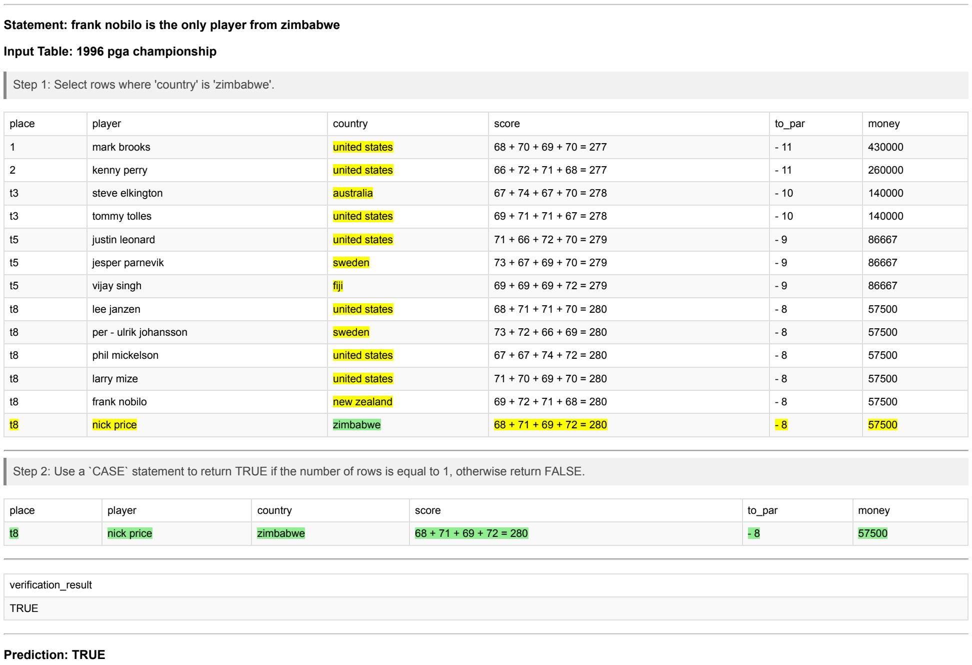
Step 2: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| place | player | country | score | to_par |
|---|---|---|---|---|
| t7 | graeme mcdowell | northern ireland | 71 + 72 = 143 | + 3 |

| verification_result |
|---|
| TRUE |

**Prediction: TRUE**

Figure 14: **POS** predicts TRUE but the groundtruth is FALSE (False Positive). In planning, **POS** misses checking the player name.



**Statement: frank nobilo is the only player from zimbabwe**

**Input Table: 1996 pga championship**

Step 1: Select rows where 'country' is 'zimbabwe'.

| place | player | country | score | to_par | money |
|---|---|---|---|---|---|
| 1 | mark brooks | united states | 68 + 70 + 69 + 70 = 277 | - 11 | 430000 |
| 2 | kenny perry | united states | 66 + 72 + 71 + 68 = 277 | - 11 | 260000 |
| t3 | steve elkington | australia | 67 + 74 + 67 + 70 = 278 | - 10 | 140000 |
| t3 | tommy tolles | united states | 69 + 71 + 71 + 67 = 278 | - 10 | 140000 |
| t5 | justin leonard | united states | 71 + 66 + 72 + 70 = 279 | - 9 | 86667 |
| t5 | jesper parnevik | sweden | 73 + 67 + 69 + 70 = 279 | - 9 | 86667 |
| t5 | vijay singh | fiji | 69 + 69 + 69 + 72 = 279 | - 9 | 86667 |
| t8 | lee janzen | united states | 68 + 71 + 71 + 70 = 280 | - 8 | 57500 |
| t8 | per - ulrik johansson | sweden | 73 + 72 + 66 + 69 = 280 | - 8 | 57500 |
| t8 | phil mickelson | united states | 67 + 67 + 74 + 72 = 280 | - 8 | 57500 |
| t8 | larry mize | united states | 71 + 70 + 69 + 70 = 280 | - 8 | 57500 |
| t8 | frank nobilo | new zealand | 69 + 72 + 71 + 68 = 280 | - 8 | 57500 |
| t8 | nick price | zimbabwe | 68 + 71 + 69 + 72 = 280 | - 8 | 57500 |

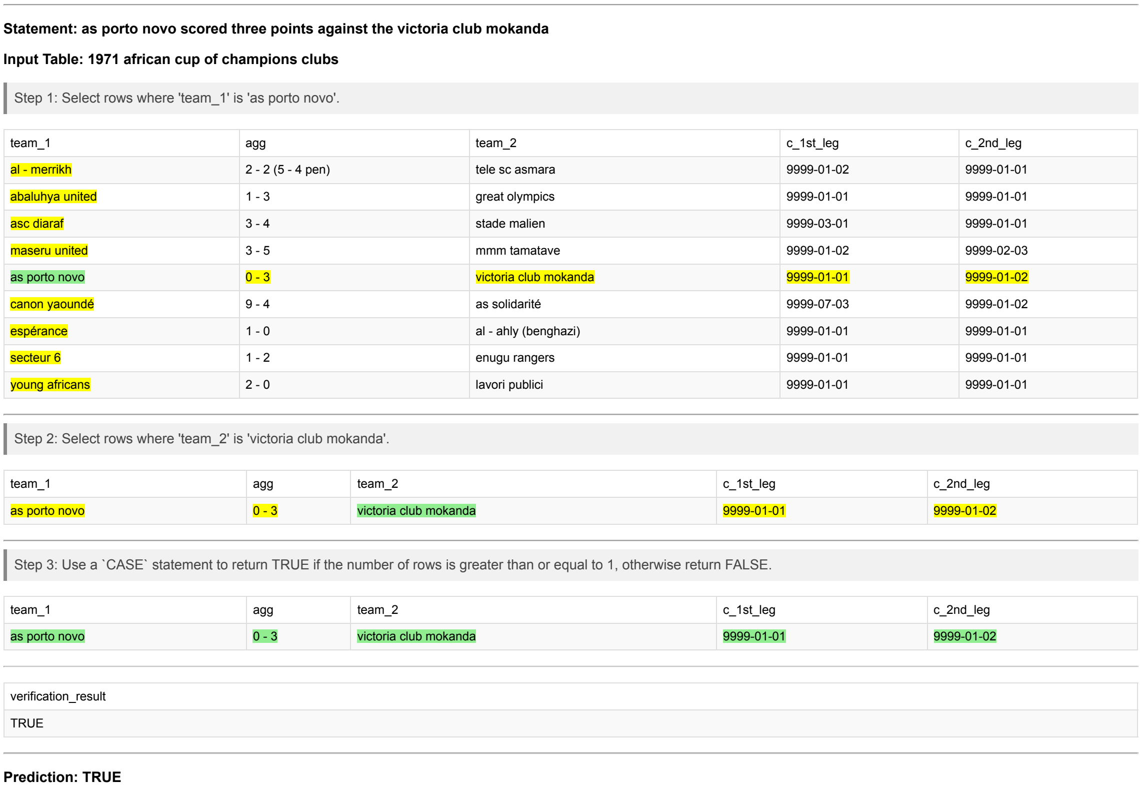
Step 2: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| place | player | country | score | to_par | money |
|---|---|---|---|---|---|
| t8 | nick price | zimbabwe | 68 + 71 + 69 + 72 = 280 | - 8 | 57500 |

| verification_result |
|---|
| TRUE |

**Prediction: TRUE**

Figure 15: `POS` predicts TRUE but the groundtruth is FALSE (False Positive). In planning, `POS` misses checking the player name.



**Statement: as porto novo scored three points against the victoria club mokanda**

**Input Table: 1971 african cup of champions clubs**

Step 1: Select rows where 'team_1' is 'as porto novo'.

| team_1 | agg | team_2 | c_1st_leg | c_2nd_leg |
|---|---|---|---|---|
| al - merrikh | 2 - 2 (5 - 4 pen) | tele sc asmara | 9999-01-02 | 9999-01-01 |
| abaluhya united | 1 - 3 | great olympics | 9999-01-01 | 9999-01-01 |
| asc diaraf | 3 - 4 | stade malien | 9999-03-01 | 9999-01-01 |
| maseru united | 3 - 5 | mmm tamatave | 9999-01-02 | 9999-02-03 |
| as porto novo | 0 - 3 | victoria club mokanda | 9999-01-01 | 9999-01-02 |
| canon yaoundé | 9 - 4 | as solidarité | 9999-07-03 | 9999-01-02 |
| espérance | 1 - 0 | al - ahly (benghazi) | 9999-01-01 | 9999-01-01 |
| secteur 6 | 1 - 2 | enugu rangers | 9999-01-01 | 9999-01-01 |
| young africans | 2 - 0 | lavori publici | 9999-01-01 | 9999-01-01 |

Step 2: Select rows where 'team_2' is 'victoria club mokanda'.

| team_1 | agg | team_2 | c_1st_leg | c_2nd_leg |
|---|---|---|---|---|
| as porto novo | 0 - 3 | victoria club mokanda | 9999-01-01 | 9999-01-02 |

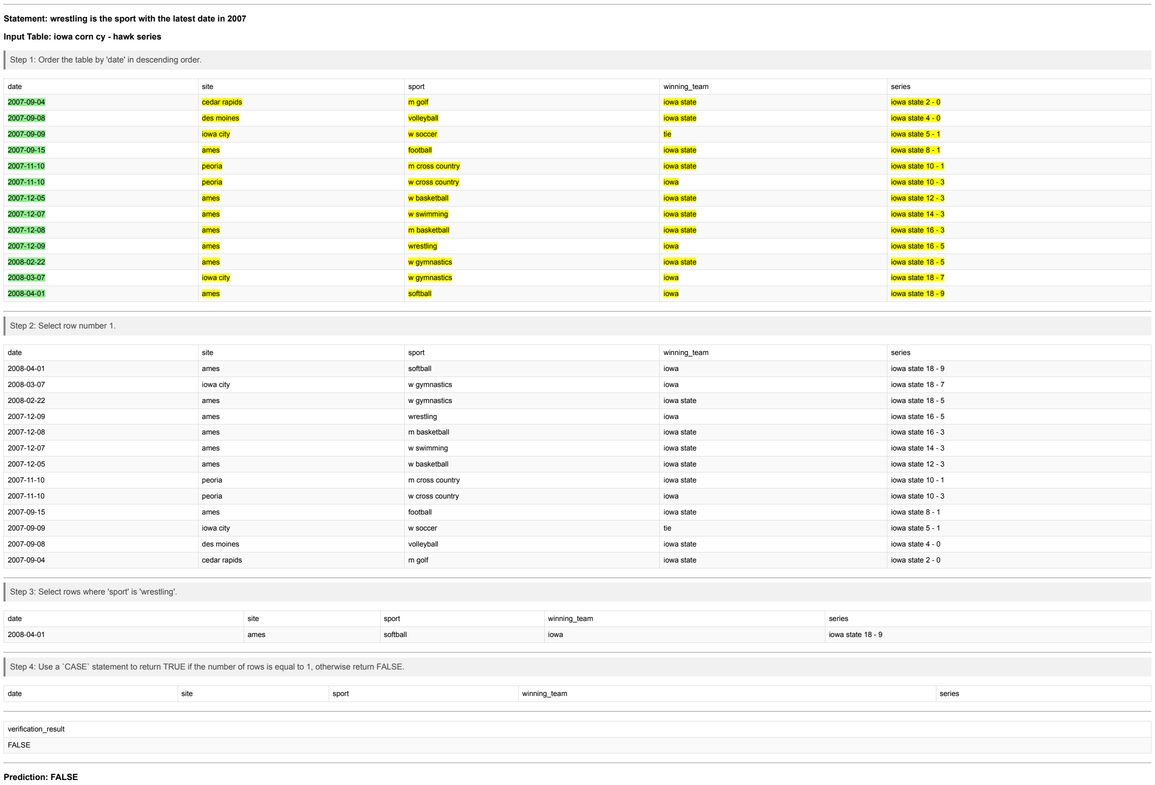
Step 3: Use a `CASE` statement to return TRUE if the number of rows is greater than or equal to 1, otherwise return FALSE.

| team_1 | agg | team_2 | c_1st_leg | c_2nd_leg |
|---|---|---|---|---|
| as porto novo | 0 - 3 | victoria club mokanda | 9999-01-01 | 9999-01-02 |

| verification_result |
|---|
| TRUE |

**Prediction: TRUE**

Figure 16: **`POS`** predicts TRUE but the groundtruth is FALSE (False Positive). In planning, **`POS`** misses checking the score.



**Statement: wrestling is the sport with the latest date in 2007**

**Input Table: iowa corn cy - hawk series**

Step 1: Order the table by 'date' in descending order.

| date | site | sport | winning_team | series |
|---|---|---|---|---|
| 2007-09-04 | cedar rapids | m golf | iowa state | iowa state 2 - 0 |
| 2007-09-08 | des moines | volleyball | iowa state | iowa state 4 - 0 |
| 2007-09-09 | iowa city | w soccer | tie | iowa state 5 - 1 |
| 2007-09-15 | ames | football | iowa state | iowa state 8 - 1 |
| 2007-11-10 | peoria | m cross country | iowa state | iowa state 10 - 1 |
| 2007-11-10 | peoria | w cross country | iowa | iowa state 10 - 3 |
| 2007-12-05 | ames | w basketball | iowa state | iowa state 12 - 3 |
| 2007-12-07 | ames | w swimming | iowa state | iowa state 14 - 3 |
| 2007-12-08 | ames | m basketball | iowa state | iowa state 16 - 3 |
| 2007-12-09 | ames | wrestling | iowa | iowa state 16 - 5 |
| 2008-02-22 | ames | w gymnastics | iowa state | iowa state 18 - 5 |
| 2008-03-07 | iowa city | w gymnastics | iowa | iowa state 18 - 7 |
| 2008-04-01 | ames | softball | iowa | iowa state 18 - 9 |

Step 2: Select row number 1.

| date | site | sport | winning_team | series |
|---|---|---|---|---|
| 2008-04-01 | ames | softball | iowa | iowa state 18 - 9 |
| 2008-03-07 | iowa city | w gymnastics | iowa | iowa state 18 - 7 |
| 2008-02-22 | ames | w gymnastics | iowa state | iowa state 18 - 5 |
| 2007-12-09 | ames | wrestling | iowa | iowa state 16 - 5 |
| 2007-12-08 | ames | m basketball | iowa state | iowa state 16 - 3 |
| 2007-12-07 | ames | w swimming | iowa state | iowa state 14 - 3 |
| 2007-12-05 | ames | w basketball | iowa state | iowa state 12 - 3 |
| 2007-11-10 | peoria | m cross country | iowa state | iowa state 10 - 1 |
| 2007-11-10 | peoria | w cross country | iowa | iowa state 10 - 3 |
| 2007-09-15 | ames | football | iowa state | iowa state 8 - 1 |
| 2007-09-09 | iowa city | w soccer | tie | iowa state 5 - 1 |
| 2007-09-08 | des moines | volleyball | iowa state | iowa state 4 - 0 |
| 2007-09-04 | cedar rapids | m golf | iowa state | iowa state 2 - 0 |

Step 3: Select rows where 'sport' is 'wrestling'.

| date | site | sport | winning_team | series |
|---|---|---|---|---|
| 2008-04-01 | ames | softball | iowa | iowa state 18 - 9 |

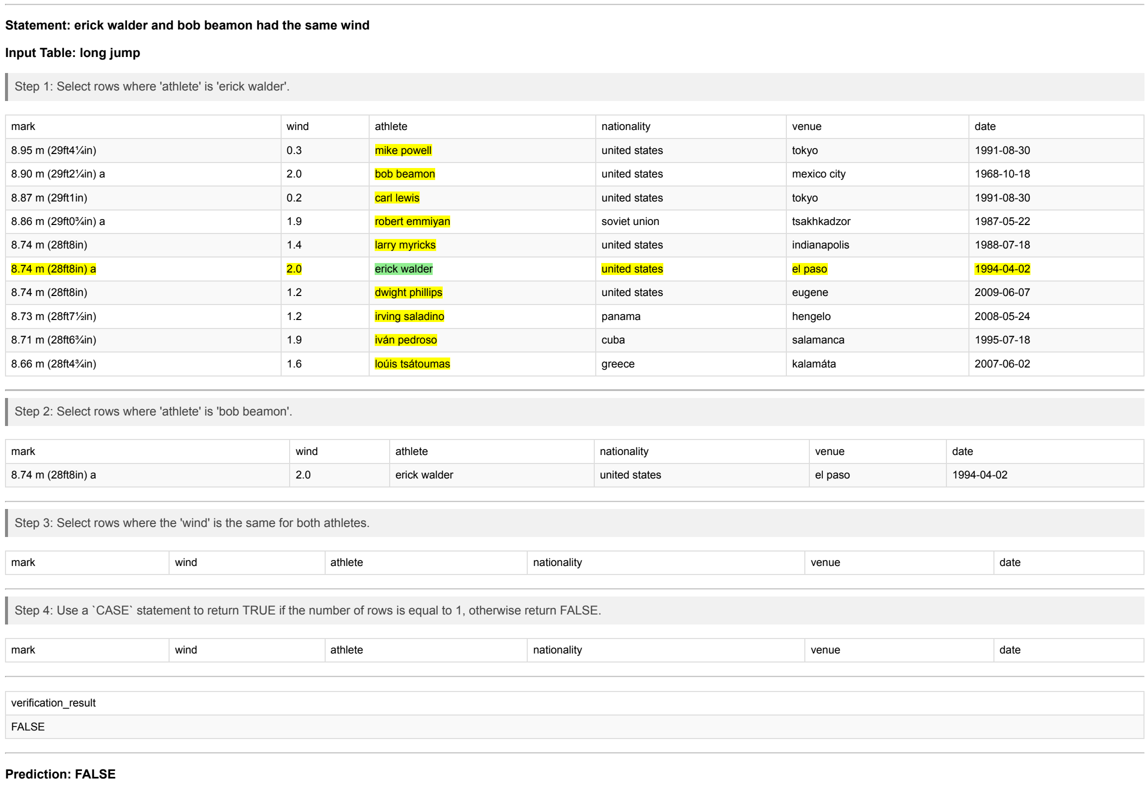
Step 4: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| date | site | sport | winning_team | series |
|---|---|---|---|---|

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

Figure 17: **`POS`** predicts FALSE but the groundtruth is TRUE (False Negative). In planning, **`POS`** misses checking the year.



**Statement: erick walder and bob beamon had the same wind**

**Input Table: long jump**

Step 1: Select rows where 'athlete' is 'erick walder'.

| mark | wind | athlete | nationality | venue | date |
|---|---|---|---|---|---|
| 8.95 m (29ft4¼in) | 0.3 | mike powell | united states | tokyo | 1991-08-30 |
| 8.90 m (29ft2¼in) a | 2.0 | bob beamon | united states | mexico city | 1968-10-18 |
| 8.87 m (29ft1in) | 0.2 | carl lewis | united states | tokyo | 1991-08-30 |
| 8.86 m (29ft0¾in) a | 1.9 | robert emmiyan | soviet union | tsakhkadzor | 1987-05-22 |
| 8.74 m (28ft8in) | 1.4 | larry myricks | united states | indianapolis | 1988-07-18 |
| 8.74 m (28ft8in) a | 2.0 | erick walder | united states | el paso | 1994-04-02 |
| 8.74 m (28ft8in) | 1.2 | dwight phillips | united states | eugene | 2009-06-07 |
| 8.73 m (28ft7½in) | 1.2 | irving saladino | panama | hengelo | 2008-05-24 |
| 8.71 m (28ft6¾in) | 1.9 | iván pedroso | cuba | salamanca | 1995-07-18 |
| 8.66 m (28ft4¾in) | 1.6 | loúis tsátoumas | greece | kalamáta | 2007-06-02 |

Step 2: Select rows where 'athlete' is 'bob beamon'.

| mark | wind | athlete | nationality | venue | date |
|---|---|---|---|---|---|
| 8.74 m (28ft8in) a | 2.0 | erick walder | united states | el paso | 1994-04-02 |

Step 3: Select rows where the 'wind' is the same for both athletes.

| mark | wind | athlete | nationality | venue | date |
|---|---|---|---|---|---|

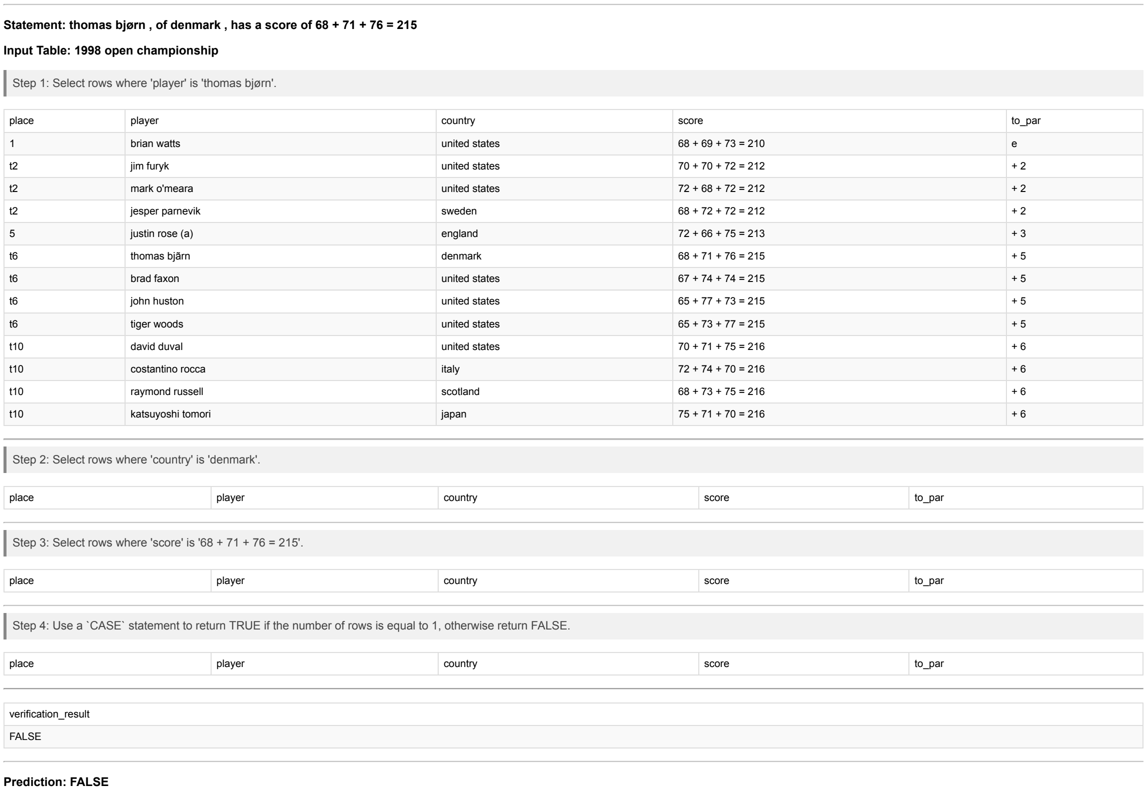
Step 4: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| mark | wind | athlete | nationality | venue | date |
|---|---|---|---|---|---|

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

Figure 18: **`POS`** predicts FALSE but the groundtruth is TRUE (False Negative). In planning, **`POS`** should select two rows at the same step.



**Statement: thomas bjørn , of denmark , has a score of 68 + 71 + 76 = 215**

**Input Table: 1998 open championship**

Step 1: Select rows where 'player' is 'thomas bjørn'.

| place | player | country | score | to_par |
|---|---|---|---|---|
| 1 | brian watts | united states | 68 + 69 + 73 = 210 | e |
| t2 | jim furyk | united states | 70 + 70 + 72 = 212 | + 2 |
| t2 | mark o'meara | united states | 72 + 68 + 72 = 212 | + 2 |
| t2 | jesper parnevik | sweden | 68 + 72 + 72 = 212 | + 2 |
| 5 | justin rose (a) | england | 72 + 66 + 75 = 213 | + 3 |
| t6 | thomas bjãrn | denmark | 68 + 71 + 76 = 215 | + 5 |
| t6 | brad faxon | united states | 67 + 74 + 74 = 215 | + 5 |
| t6 | john huston | united states | 65 + 77 + 73 = 215 | + 5 |
| t6 | tiger woods | united states | 65 + 73 + 77 = 215 | + 5 |
| t10 | david duval | united states | 70 + 71 + 75 = 216 | + 6 |
| t10 | costantino rocca | italy | 72 + 74 + 70 = 216 | + 6 |
| t10 | raymond russell | scotland | 68 + 73 + 75 = 216 | + 6 |
| t10 | katsuyoshi tomori | japan | 75 + 71 + 70 = 216 | + 6 |

Step 2: Select rows where 'country' is 'denmark'.

| place | player | country | score | to_par |
|---|---|---|---|---|

Step 3: Select rows where 'score' is '68 + 71 + 76 = 215'.

| place | player | country | score | to_par |
|---|---|---|---|---|

Step 4: Use a `CASE` statement to return TRUE if the number of rows is equal to 1, otherwise return FALSE.

| place | player | country | score | to_par |
|---|---|---|---|---|

| verification_result |
|---|
| FALSE |

**Prediction: FALSE**

Figure 19: **`POS`** predicts FALSE but the groundtruth is TRUE (False Negative). The exact-matching nature of SQL makes **`POS`** cannot retrieve the relevant information.

## L Human Study Interface

To evaluate the effectiveness of different explanation methods in Table QA on human users, we develop a web-based interface using HuggingFace Gradio and Flask. The interface is designed for Forward Simulation to guide participants through the study seamlessly, ensuring they understand the tasks and provide valuable feedback.

Overview of the Forward Simulation Interface Flow:

1. Informed Consent ⇒
2. Introduction to Table QA and Forward Simulation ⇒
3. Introduction to Explanations in Table QA ⇒
4. Welcome page where users are asked to choose one of 4 XAI methods ⇒
5. Specific explanation page for the chosen method ⇒
6. Experiment pages for 10 samples ⇒
7. Completion page!

### L.1 Informed Consent

## Informed Consent for Table QA Study

**Study Information:**
You are invited to participate in a research study on Table QA systems. This study aims to improve how AI systems explain their reasoning when answering questions based on tabular data.

**Study Duration and Requirements:**

1. The entire study will take approximately **20 minutes** to complete.
2. Please perform this study on a **computer** (not a phone).
3. Do not seek help from the Internet or other people during the study.

**Study Structure:**

1. Introduction to Table QA and task explanation
2. Main study: 10 questions about Table QA explanations

**Benefits:**
Your participation will contribute to the development of AI systems that can better explain their reasoning to humans, particularly in the domain of question answering from tabular data. There are no known risks associated with this study.

**Data Usage and Confidentiality:**
All data collected will be anonymized and used solely for research purposes. Your personal information will be kept confidential.

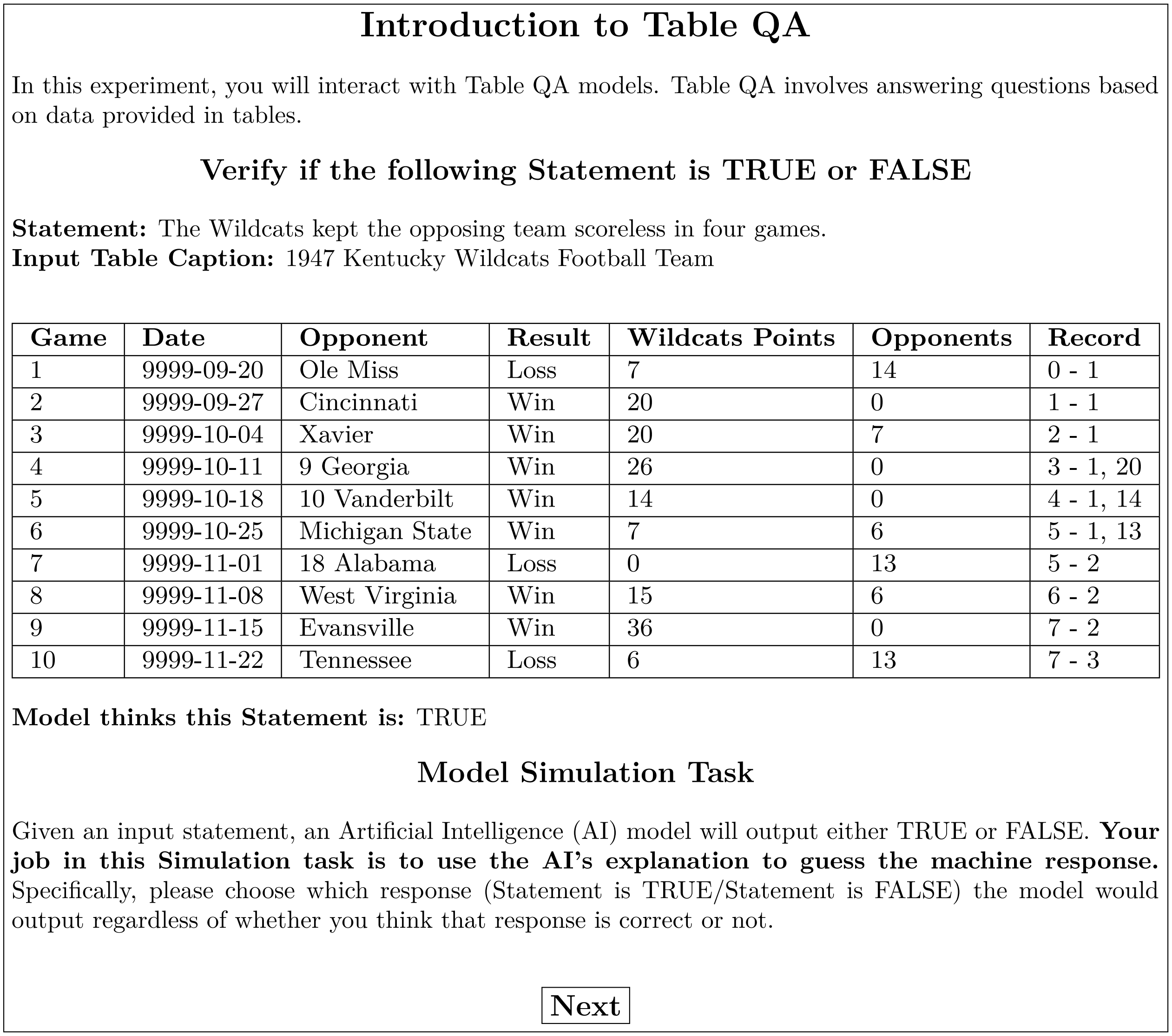
**Voluntary Participation:**
Your participation in this study is entirely voluntary. You may choose to withdraw at any time without any consequences.

**Contact Information:**
If you have any questions or concerns about this study, please contact [anonymized].

**Agreement:**
By clicking "**I Agree**" below, you confirm that you have read and understood this informed consent, and you agree to participate in this Table QA study under the terms described above.

**I Agree**

### L.2 Introduction to Table QA and Forward Simulation



## Introduction to Table QA

In this experiment, you will interact with Table QA models. Table QA involves answering questions based on data provided in tables.

### Verify if the following Statement is TRUE or FALSE

**Statement:** The Wildcats kept the opposing team scoreless in four games.
**Input Table Caption:** 1947 Kentucky Wildcats Football Team

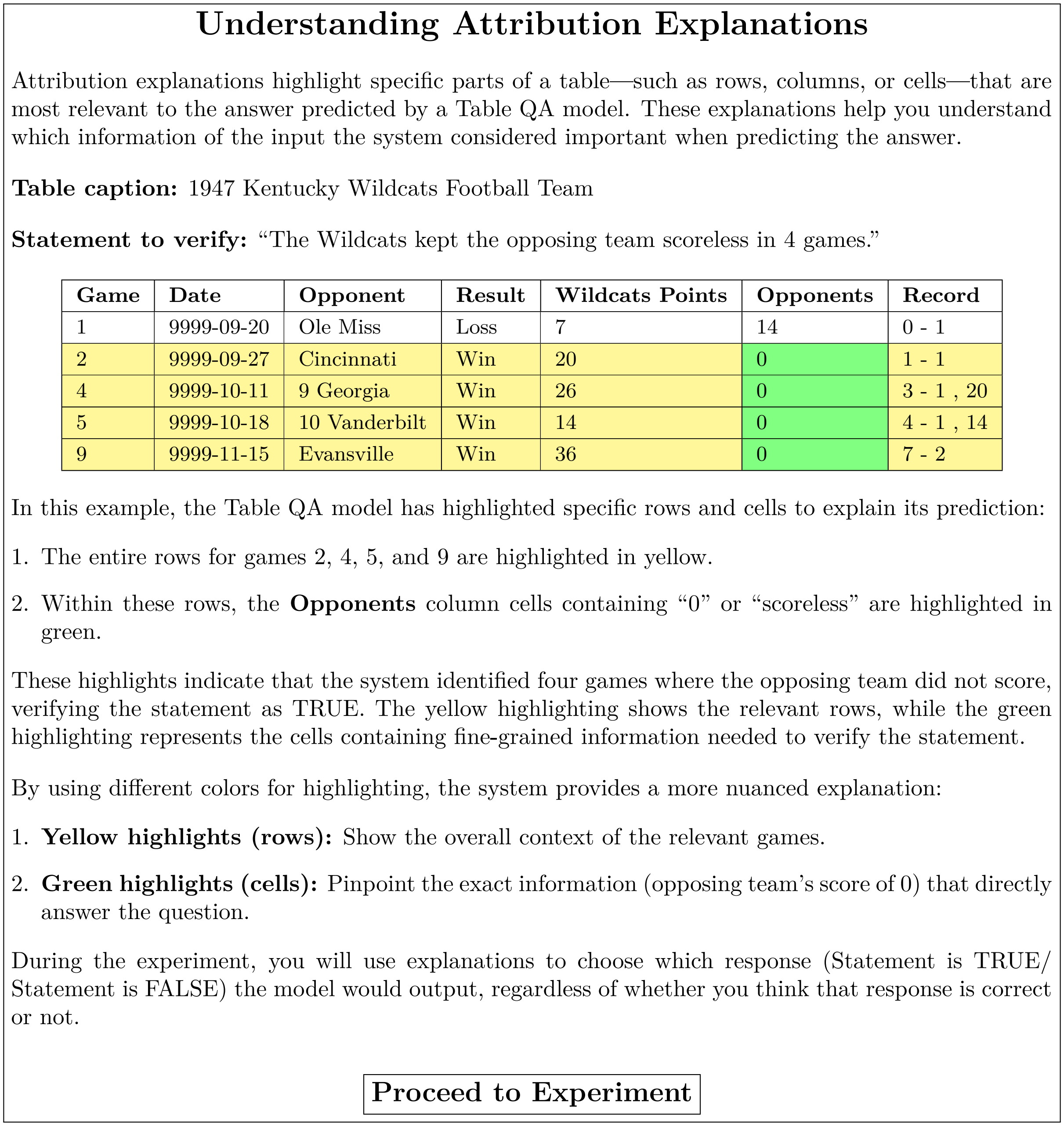
| Game | Date | Opponent | Result | Wildcats Points | Opponents | Record |
|---|---|---|---|---|---|---|
| 1 | 9999-09-20 | Ole Miss | Loss | 7 | 14 | 0 - 1 |
| 2 | 9999-09-27 | Cincinnati | Win | 20 | 0 | 1 - 1 |
| 3 | 9999-10-04 | Xavier | Win | 20 | 7 | 2 - 1 |
| 4 | 9999-10-11 | 9 Georgia | Win | 26 | 0 | 3 - 1, 20 |
| 5 | 9999-10-18 | 10 Vanderbilt | Win | 14 | 0 | 4 - 1, 14 |
| 6 | 9999-10-25 | Michigan State | Win | 7 | 6 | 5 - 1, 13 |
| 7 | 9999-11-01 | 18 Alabama | Loss | 0 | 13 | 5 - 2 |
| 8 | 9999-11-08 | West Virginia | Win | 15 | 6 | 6 - 2 |
| 9 | 9999-11-15 | Evansville | Win | 36 | 0 | 7 - 2 |
| 10 | 9999-11-22 | Tennessee | Loss | 6 | 13 | 7 - 3 |

**Model thinks this Statement is:** TRUE

### Model Simulation Task

Given an input statement, an Artificial Intelligence (AI) model will output either TRUE or FALSE. **Your job in this Simulation task is to use the AI's explanation to guess the machine response.** Specifically, please choose which response (Statement is TRUE/Statement is FALSE) the model would output regardless of whether you think that response is correct or not.

### L.3 Introduction to Explanations in Table QA



## Understanding Attribution Explanations

Attribution explanations highlight specific parts of a table—such as rows, columns, or cells—that are most relevant to the answer predicted by a Table QA model. These explanations help you understand which information of the input the system considered important when predicting the answer.

**Table caption:** 1947 Kentucky Wildcats Football Team

**Statement to verify:** "The Wildcats kept the opposing team scoreless in 4 games."

| Game | Date | Opponent | Result | Wildcats Points | Opponents | Record |
|---|---|---|---|---|---|---|
| 1 | 9999-09-20 | Ole Miss | Loss | 7 | 14 | 0 - 1 |
| 2 | 9999-09-27 | Cincinnati | Win | 20 | 0 | 1 - 1 |
| 4 | 9999-10-11 | 9 Georgia | Win | 26 | 0 | 3 - 1 , 20 |
| 5 | 9999-10-18 | 10 Vanderbilt | Win | 14 | 0 | 4 - 1 , 14 |
| 9 | 9999-11-15 | Evansville | Win | 36 | 0 | 7 - 2 |

In this example, the Table QA model has highlighted specific rows and cells to explain its prediction:

1. The entire rows for games 2, 4, 5, and 9 are highlighted in yellow.
2. Within these rows, the **Opponents** column cells containing "0" or "scoreless" are highlighted in green.

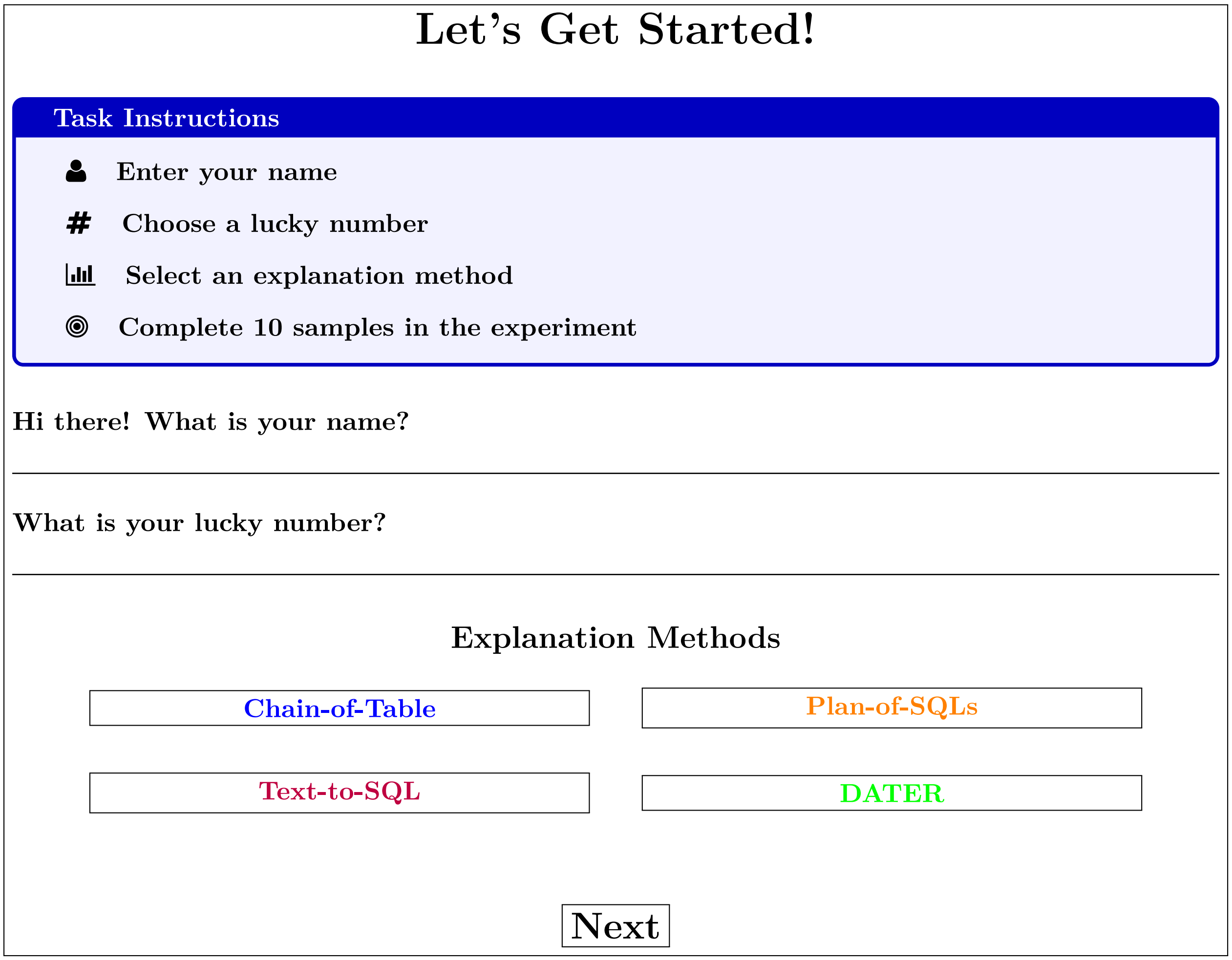
These highlights indicate that the system identified four games where the opposing team did not score, verifying the statement as TRUE. The yellow highlighting shows the relevant rows, while the green highlighting represents the cells containing fine-grained information needed to verify the statement.

By using different colors for highlighting, the system provides a more nuanced explanation:

1. **Yellow highlights (rows):** Show the overall context of the relevant games.
2. **Green highlights (cells):** Pinpoint the exact information (opposing team's score of 0) that directly answer the question.

During the experiment, you will use explanations to choose which response (Statement is TRUE/ Statement is FALSE) the model would output, regardless of whether you think that response is correct or not.

### L.4 Welcome Page



# Let's Get Started!

**Task Instructions**

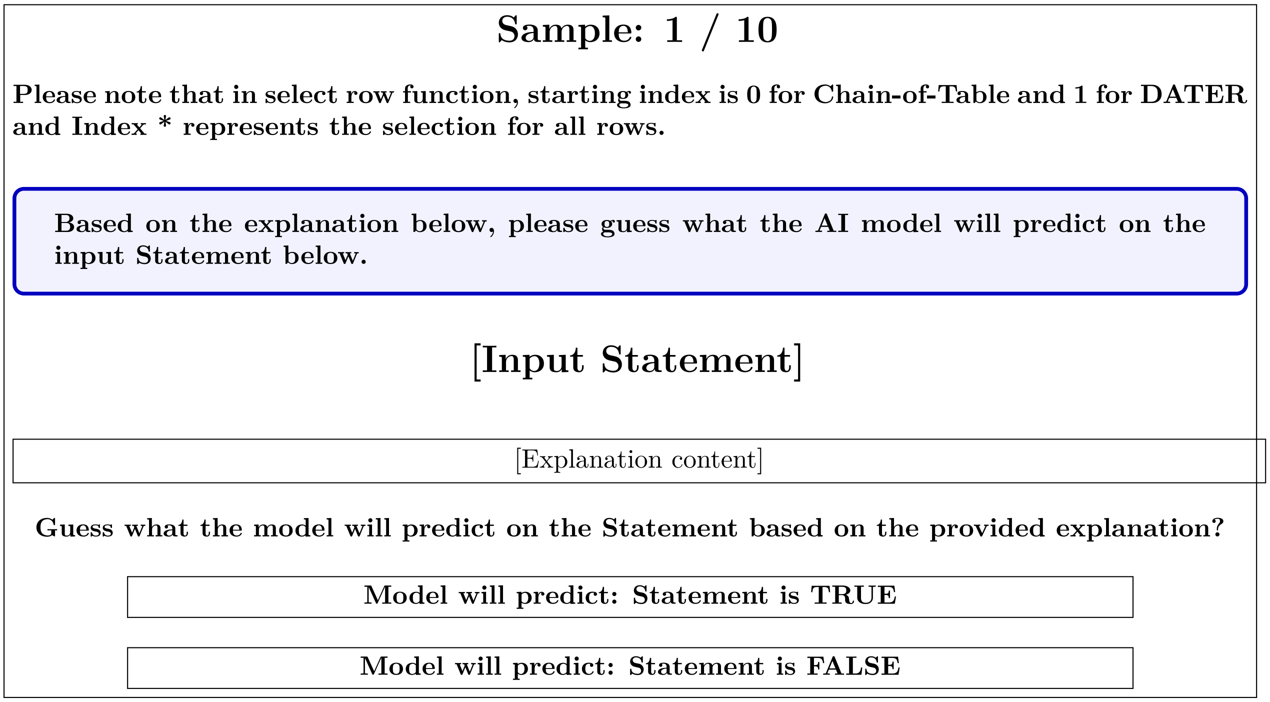
- 👤 Enter your name
- # Choose a lucky number
- 📊 Select an explanation method
- ◎ Complete 10 samples in the experiment

**Hi there! What is your name?**

**What is your lucky number?**

Explanation Methods

### L.5 Experiment Page



## Sample: 1 / 10

**Please note that in select row function, starting index is 0 for Chain-of-Table and 1 for DATER and Index * represents the selection for all rows.**

**Based on the explanation below, please guess what the AI model will predict on the input Statement below.**

## [Input Statement]

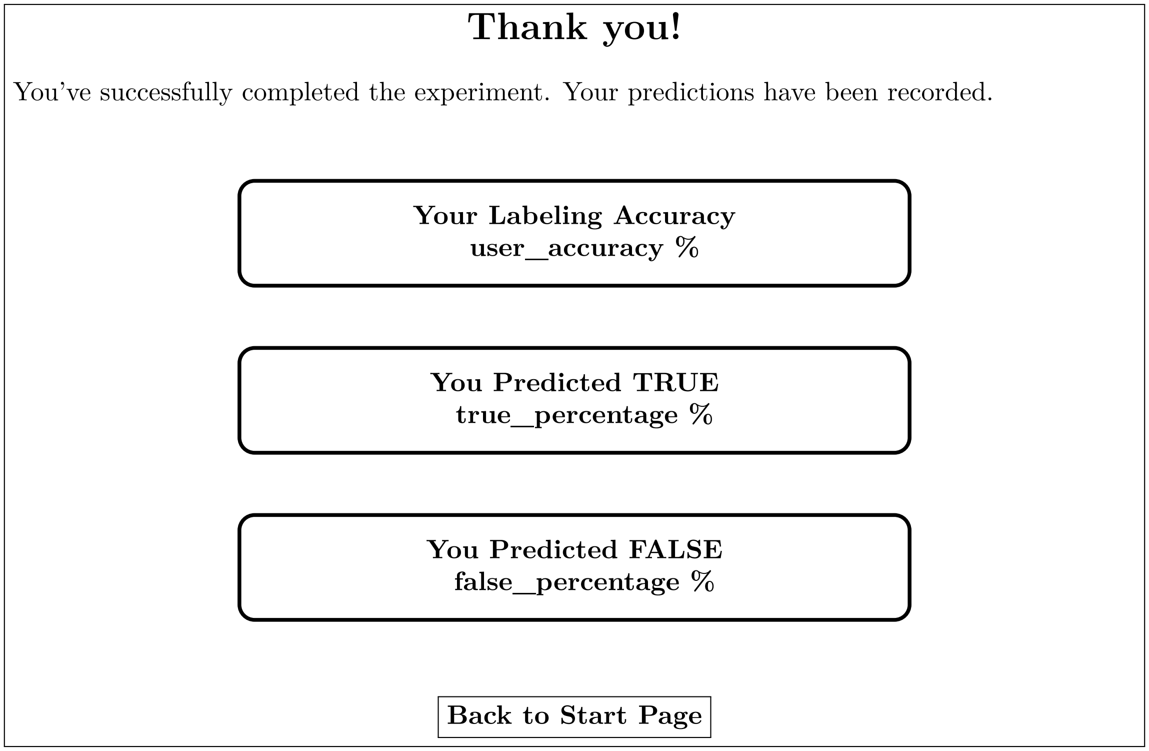
[Explanation content]

**Guess what the model will predict on the Statement based on the provided explanation?**

**Model will predict: Statement is TRUE**

**Model will predict: Statement is FALSE**

### L.6 Completion Page



## Thank you!

You've successfully completed the experiment. Your predictions have been recorded.

**Your Labeling Accuracy**
**user_accuracy %**

**You Predicted TRUE**
**true_percentage %**

**You Predicted FALSE**
**false_percentage %**

**Back to Start Page**